Qualitative Multiple-Fault Diagnosis of Continuous Dynamic Systems Using Behavioral Modes

by

Siddarth Subramanian, B. Tech.,M.S.C.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 1995

Qualitative Multiple-Fault Diagnosis of Continuous Dynamic Systems Using Behavioral Modes

Approved by
Dissertation Committee:

_____

_____

_____

_____

_____

_____

To my parents

Acknowledgments

I would first like to thank my adviser, Ray Mooney, for all his help with my work, the encouragement he has given me over years of ups and downs in my research, and for his patience and perseverance in reading through repeated revisions of this dissertation.

The granting agencies which have partially supported this research include the National Science Foundation (under grants IRI-9310819 and IRI-9102926), Texas Advanced Research Projects (grant ARP-003658-114) and the National Aeronautical and Space Administration Ames Research Center (grant NCC 2-629). The financial help I received from them was instrumental in enabling me to continue in graduate school and finish the dissertation.

The students and faculty in the department who helped in some way are too numerous to list completely. I am listing the most important ones and hope that any omissions are forgiven. First, the members of my committee who gave me many helpful comments and contributed to the revisions were (in no particular order) Peter Clark, Lyle Ungar, and Ben Kuipers. I thank them all and hope this revised document meets with their approval. A special thank you goes to Peter Clark for helping provide a new perspective on my results.

The student who perhaps had the most significant impact on my work in these last two years of graduate school was Bert Kay. I thank him for letting me barge into his office without warning, for his many helpful suggestions, and in particular, for the detailed help he provided on the model of the Reaction Control System.

Other students who have helped significantly in recent years include Jeff Rickel, Rich Mallory, Sowmya Ramachandran, and the THREV group consisting of Jeff Mahoney, John Zelle, Cindi Thompson, Tara Estlin, Paul Baffes, and Mary Elaine Califf. Others who deserve special mentions for their friendship and camaraderie over the years include Tim Collins, Neelakantan Kartha, and James Lester.

My parents continued to believe in me through the years. For this, and for their continued support, I thank them.

Finally, I would like to thank my future wife Cilla McMillen (whom I intend to marry on August 25th, just a week after I submit this dissertation). With her love and support I found the will to keep on working when I would have preferred to give up. I owe her the

greatest debt of gratitude.

Siddarth Subramanian

The University of Texas at Austin
December 1995

Qualitative Multiple-Fault Diagnosis of Continuous Dynamic Systems Using Behavioral Modes

Publication No. _____

Siddarth Subramanian, Ph.D.
The University of Texas at Austin, 1995

Supervisor: Raymond J. Mooney

As systems like chemical plants, power plants, and automobiles get more complex, online diagnostic systems are becoming increasingly important. One of the ways to rein in the complexity of describing and reasoning about large systems such as these is to describe them using qualitative rather than quantitative models.

Model-based diagnosis is a class of diagnostic techniques that use direct knowledge about how a system functions instead of expert rules detailing causes for every possible set of symptoms of a broken system. Our research builds on standard methods for model-based diagnosis and extends them to the domain of complex dynamic systems described using qualitative models.

We motivate and describe our algorithm for diagnosing faults in a dynamic system given a qualitative model and a sequence of qualitative states. The main contributions in this algorithm include a method for propagating dependencies while solving a general constraint satisfaction problem, and a method for verifying the compatibility of a behavior with a model across time. The algorithm can diagnose multiple faults and uses models of faulty behavior, or behavioral modes.

We then demonstrate these techniques using an implemented program called QDOCS and test it on some realistic problems. Through our experiments with a model of the reaction control system (RCS) of the space shuttle and with a level-controller for a reaction tank, we show that QDOCS demonstrates the best balance of generality, accuracy and efficiency among known systems.

Contents

Chapter 1

Introduction

1.1 The Case for Automated Diagnosis

The world is increasingly filled with complex devices. Although systems like automobiles, aircraft, chemical plants, and others, have had the same basic design for many decades, various subsystems have been added to meet the need for increased efficiency, increased power, and to meet design criteria such as safety and emission standards.

This trend of complication in devices has also meant that they are more and more likely to fail in unpredictable ways. When they do, few human users understand enough about these systems to be able to instantly diagnose the problem. Since unchecked failures in such devices can sometimes have catastrophic effects, it is important to develop automated monitoring systems that are able to diagnose faults as soon as they are detectable.

Various diagnosis algorithms have been devised, and diagnostic systems have been built, over the last two decades [Shortliffe & Buchanan, 1975; de Kleer & Williams, 1987; Reggia, Nau, & Wang, 1983]. These systems have been applied to the problems of medical diagnosis, as well as to combinational circuit diagnosis and similar domains. However, as we shall see, none of these diagnosis approaches are suitable for the kinds of continuous dynamic systems that we are interested in.

The problem of building an effective diagnosis system that works in real-time can be broken up into two subproblems. The first, known as *monitoring*, involves extracting enough information from the device to determine the state of the device at any given time. A monitoring system must also be able to detect that the device is in an anomalous state. The second, which is the task more commonly thought of as diagnosis, is that of determining what components could have "broken" and in what way they may have broken. It is this second task that this research seeks to address in the context of complex dynamic systems.

1.2 The Case for Qualitative Reasoning

Complex devices have complex dynamics. Differential equation models can be built for most simple devices and these can often be chained together to model many complex devices. We also have numerical methods for solving almost any differential equation. However, these methods get prohibitively expensive when reasoning about large differential equation models with hundreds of variables.

Furthermore, these differential equation models are often too difficult to formulate precisely for all systems, as they often depend on knowledge of the exact parameters and dimensions of the system and thus require precise measurements of all these parameters.

Somehow, humans are able to reason about physical systems without such detailed knowledge. We are able to imagine what happens when, say, a car's radiator starts slowly leaking – we are able to reason through the fact that there must be less water passing through the engine block, to be able to conclude that the car will overheat. Conversely, if we see the car overheating, we can easily come up with the diagnostic hypothesis that the radiator is leaking, using the same chain of reasoning backwards.

The reason we are able to arrive at such conclusions is because we have the ability to reason *qualitatively* about physical systems. By qualitative reasoning, we mean we can reason in terms of levels, pressures, flows, etc., as being increasing or decreasing or above or below some *landmark* value, without having to know precisely what their values are. Instead of thinking in terms of differential equations, we propagate effects of values around a mental model of the device, by using our knowledge of how different values affect others.

Given that the devices we wish to diagnose faults in are complex, dynamic devices, the above problems with building and simulating differential equation models apply. This research therefore focusses on attempting to use qualitative techniques to solve the diagnosis problem for complex dynamic systems.

A number of approaches to the problem of modeling systems qualitatively have been studied in the literature [deKleer & Brown, 1984; Forbus, 1984; Kuipers, 1984]. These modeling techniques, however, have mainly been studied in the context of simulation systems. While there have been some diagnosis systems to come out of this area of research (e.g., [Ng, 1990], [Dvorak, 1992] and others), these solutions to the diagnosis problem are, as we shall see, limited in nature.

1.3 Diagnosis for Systems Modeled Qualitatively

The research we report on in this dissertation is an exploration of applying automated diagnosis techniques to dynamic systems modeled qualitatively. In presenting our work, we will highlight some key distinctions between our approach, and those of previously published research. Our main contributions can be enumerated as follows.

1. We take a very general approach to the problem by combining a multiple-fault, fault-model based diagnosis algorithm and a general qualitative language to describe continuous dynamic systems.

2. We develop techniques for propagating dependencies in the context of solving constraint satisfaction problems and we use these to extract information useful to our diagnosis system.

3. We report on empirical tests of our algorithms obtained by applying them to large continous dynamic systems in two different domains, and show that they offer practical benefits over competing methods.

We will first introduce some of the early research in diagnosis and qualitative reasoning, as well as some of the earlier research on the qualitative diagnosis of dynamic systems (Chapter 2). Next, in Chapter 3, we present the algorithms we use in our system QDOCS (for Qualitative Diagnosis Of Continuous Systems). This is followed, in Chapter 4, by a look at some of the ways in which we have extended our algorithms to attempt to make them more general or more efficient. Our empirical testing and evaluation of QDOCS appears in Chapter 5 where we apply our system to the diagnosis of faults in the Reaction Control System of the Space Shuttle, as well as to a level-controlled reaction tank.

Chapter 6 then compares QDOCS to some of the other diagnostic methods that have been applied to dynamic systems. In Chapter 7 we present some suggestions for future directions for this research, and finally, in Chapter 8 we give our conclusions.

Chapter 2

Background

This chapter provides an introduction to two of the main research areas that are integrated in this dissertation. These areas, Model-Based Diagnosis and Qualitative Reasoning, have developed essentially independently even though some of the main researchers in each area have also done work in the other. We follow this with a discussion of some of the early work on qualitative diagnosis of dynamic systems. This will set the stage and motivate our own approach to the problem.

2.1 Model-Based Diagnosis

2.1.1 Classifying Model-Based Diagnostic Systems

The task of diagnosis is loosely defined as the reasoning that explains the symptoms exhibited by a particular system, by inferring the causes for those symptoms. In the last two decades, diagnostic reasoning has been an important focus of AI research. Starting with Mycin[Shortliffe & Buchanan, 1975], a program to diagnose infectious diseases, a number of systems have utilized an expert systems approach to diagnosis. The approach in these systems was to codify expert knowledge as rules that reasoned from symptoms to the suspected underlying causes.

Figure 2.1 shows a diagram of the full adder circuit that is commonly used to demonstrate and test diagnosis algorithms. The circuit normally works by adding the two bits in `b1` and `b2` and then adding in the carry in `c`. The two outputs are the output bit `bit-out` and the carry bit `carry-out`.
Suppose, at some point, the inputs are

`b1` $= 1$, `b2` $= 0$, and `c` $= 1$.

The expected outputs are

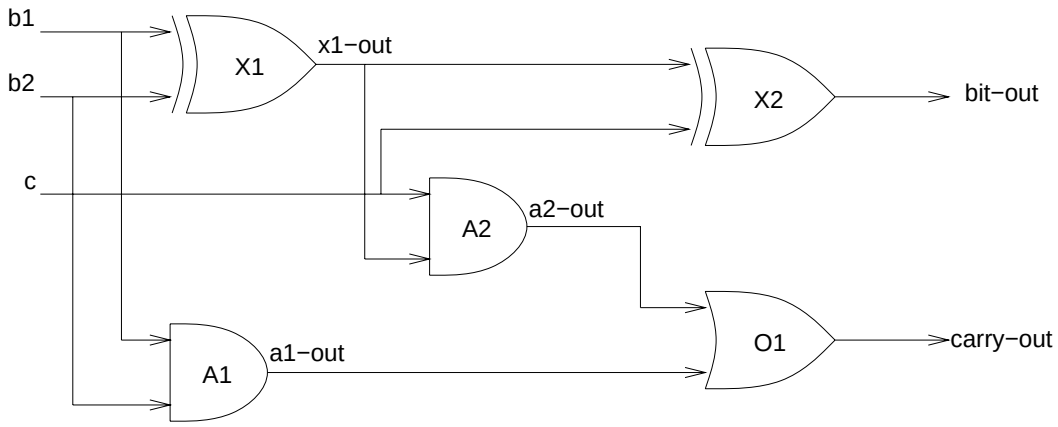

Figure 2.1: Full Adder Circuit

`bit-out` $= 0$ and `carry-out` $= 1$.

If instead `carry-out` equals 0, while `bit-out` is also 0 as expected, we know that there is some fault in the system. Some of the possible faults are that the or-gate `o1` is stuck at 0, or that the and-gate `a1` is stuck at 0. Alternatively, `x1` and `x2` could both be stuck at 0.

The expert systems approach to the problem of diagnosis in this domain would be for an expert to exhaustively list rules for determining what the faults could be given particular outputs. So, in this case, there would have to be rules suggesting all of the above possibilities. Similar rules would exist for each distinct set of anomalous outputs for each set of inputs.

In the mid-1980's, however, it was recognized by various researchers ([Reggia et al., 1983],[Davis, 1984], and others) that this approach suffered from a knowledge acquisition problem. In other words, it would take a lot of work to distill all of an expert's knowledge of a device or system and express it in terms of symptom-cause rules.

Such sets of rules are also hard to update to similar devices. If we added another gate somewhere in the circuit of Figure 2.1, we would have to rebuild the entire set of rules for the device. Further, to be complete, the rules would have to suggest all single-fault possibilities for a set of anomalous outputs, as well as all the double-faults, triple-faults, etc. Because of these problems, most such rule-based systems, like the ones from medical diagnosis, were limited to heuristic rules that would suggest the most probable diagnoses given a set of symptoms, and leave out some of the more marginal possibilities.

Researchers such as Reggia and Davis realized that human experts had knowledge of an underlying causal model of the device that they used to reason from symptoms to diagnoses. Since the underlying model was what the expert used, these researchers argued that this model was easier to elicit from the expert than symptom-cause rules. The under-

lying models would also be easier to update because, for example, experts have no problem moving from an analysis of a given circuit to one with one additional gate.

This realization led to the body of work that is known as Model-Based Diagnosis (MBD) [Hamscher, Console, & deKleer, 1992]. An MBD system is one that starts with some sort of causal model of how a device operates and uses this model to diagnose causes when the symptoms are given.

MBD systems are further subdivided into a number of different types. Poole [1989] categorizes such systems along two axes. The first is the distinction between *consistency-based* and *abductive* diagnostic systems. The distinction here is on what the diagnostic systems are actually trying to achieve. In a consistency-based system, the effort is to find some characterization of the device that is consistent with the observed symptoms. An abductive system, on the other hand, is more concerned with actually explaining the observed symptoms causally.

The distinction here is subtle but important. In the case of our adder circuit, and given the inputs and outputs mentioned above, a consistency-based diagnostic system may diagnose the problem as simply being in the or-gate `o1`. There need not be an accounting of why the outputs are what they are in a consistency-based system. Instead, if simply denying that the or-gate is working correctly makes the inputs and outputs consistent, then this hypothesis is a consistency-based diagnosis of the observations. Thus the model of the device would simply need to have information on how the device is supposed to function if all the components are functioning normally.

However, for an abductive system (e.g., [Reggia et al., 1983]), a satisfactory diagnosis would be one where the assumptions logically imply the observed behavior. So, given our inputs and outputs above, we would have to make the assumption that the or-gate `o1` is stuck at zero and have logical rules that would then explain the value of 0 at the or-gate.

An alternative axis along which we can characterize diagnostic systems is according to whether the hypotheses they generate consist of *behavioral modes* of components of a device or if they simply indicate that a component may be broken. Behavioral modes are known faulty, as well, as normal ways in which a component may be operating. Thus for an or-gate such as `o1`, some of the possible modes may include `normal` and `stuck-at-0`. A diagnosis system that gives the behavioral modes of each component of a device under faulty observations is sometimes called a *fault-based* diagnosis system, as opposed to a *normality-based* system which simply determines which components may be broken.

It may appear at first sight that normality-based diagnosis would have to be consistency-based and fault-based would have to be abductive. In fact, most of the work on abductive diagnosis ([Reggia et al., 1983],[Ng, 1992], etc.) was also fault-based while most early consistency-based systems ([Reiter, 1987; de Kleer & Williams, 1987]) were normality based. However, there is no reason why a system should not be both fault-based and consistency-

1. Propagate input values through constraints to make predictions to output values. Using a trace of the constraint propagation, determine, for each predicted output value, the components that its value is dependent on. This is the *dependency set* for this predicted output value.
2. Consider all the output values which are different from their predicted values. Take the intersection of the dependency sets for each of these values to determine the initial set of candidates.
3. For each candidate in this set, suspend the constraints associated with the candidate, and use constraint propagation to determine consistency of the candidate with the observations.

Figure 2.2: Constraint Suspension algorithm

based and, in fact, Poole [1989] shows that these are orthogonal axes of classification. de Kleer and Williams [1989] and Struss and Dressler [1989] both independently formulated algorithms for consistency-based diagnosis with behavioral modes.[1]

A consistency-based system with behavioral modes differs from an abductive system in that it only needs to establish that some assignment of behavioral modes to components is consistent with the observations. An abductive system using behavioral modes would actually have to explain the observations.

For reasons that we shall give later, our research follows the consistency-based and fault-based modes of diagnostic reasoning, and our main diagnostic engine is similar to SHERLOCK, the algorithm introduced in [de Kleer & Williams, 1989]. In the following sections, we introduce and discuss the precursors of this algorithm followed by the SHERLOCK algorithm itself.

2.1.2 Constraint Suspension

Constraint Suspension is one of the earlier techniques used for consistency-based diagnosis [Davis, 1984]. As in many other constraint-based systems we shall be looking at, the algorithm requires components of the device to be associated with constraints. Predictions are made using a technique called *constraint propagation*, where individual constraints are used to determine values of variables they act upon, based on the values of other variables they act upon.

The technique of constraint suspension is a simple one and is summarized in Figure 2.2. Note from step 2 that this method makes a *single-fault assumption*. This is a fairly common assumption in diagnosis systems, but one that is often not warranted, since normally independent components may fail simultaneously, either by coincidence, or because of the same external factors.

[1] We shall use the terms *fault modes* and *behavioral modes* interchangably. The latter term is used to refer to all the fault modes as well as the normal modes of the component.

Constraint suspension works simply by tracking through the constraint propagation to determine which components could possibly have affected a given output value. It then takes the intersection of all such dependency sets corresponding to discrepant predictions and then allows all the constraints associated with one component at a time to be suspended. For each such suspended set of constraints, the consistency of the device is checked to determine if the component is a possible diagnosis.

In the example of Figure 2.1, if `b1, b2,` and `c1` had values of 1, 0, and 1, respectively, and if `bit-out` and `carry-out` were 0 and 0 respectively, then the only anomalous output would be the `carry-out` value of 0. A dependency trace would reveal that the components `x1`, `o1`, and `a2` could have been involved in determining the anomalous value for `carry-out`.

The procedure would then have to suspend the constraints associated with each of these and determine if the resulting model was now consistent with the observations. In this case, all the candidates are consistent and could be considered possible diagnoses.

The main problem with this method is that the single-fault assumption is too restrictive and does not account for cases where different components could fail simultaneously. In fact, in cases which can only be covered using multiple-fault hypotheses, the method simply fails completely since candidate hypotheses need not be limited to the intersection of the dependency sets on the anomalous outputs.

2.1.3 Reiter's Theory of Diagnosis and GDE

Reiter [1987] was one of the first to propose a formal theory of diagnosis. His work also includes an algorithm that is the basis for the diagnostic algorithm in QDOCS. Since much of the terminology he uses is standard in model-based diagnosis and shall be used throughout this dissertation, we introduce them now.

The following is a definition of the diagnostic problem.

Definition 2.1.1 (Reiter) *A* diagnosis problem *consists of a triple* (SD, COMPONENTS, OBS) *where* SD *is a system description,* COMPONENTS *is a set of components, and* OBS *is a set of observations.*

In Reiter's formulation, the system description is a set of first-order formulae, the components are a set of constants and the observations are another set of formulae. The full adder of Figure 2.1 would have a system description that would include formulae like the following:

$$(\texttt{andg(a)} \wedge (\texttt{in1(a)} = 1) \wedge (\texttt{in2(a)} = 1) \wedge \neg \text{AB}(\texttt{a}) \longrightarrow (\texttt{out(a)} = 1)$$

$$(\texttt{andg(a)} \wedge ((\texttt{in1(a)} = 0) \vee (\texttt{in2(a)} = 0)) \wedge \neg \text{AB}(\texttt{a}) \longrightarrow (\texttt{out(a)} = 0)$$

In other words, if the AB predicate does not have a true value for the and-gate `a`, then the value for `out(a)` is given by the conjunction of `in1(a)` and `in2(a)` (where true = 1 and

false = 0). The system description also contains similar formulae for each of the types of components in the system and circuit-specific formulae such as `andg(a1)`, `andg(a2)`, etc. The OBSERVATIONS here are all the input and the output values in any given situation, e.g.,

$$(\texttt{b1} = 1) \ \wedge \ (\texttt{b2} = 0) \ \wedge \ (\texttt{c1} = 1) \ \wedge (\texttt{bit-out} = 0) \ \wedge \ (\texttt{carry-out} = 0)$$

AB is thus a predicate that must be defined over all the components of the system. Its intuitive meaning is that if it holds true for any component, that component is believed to be faulty and thus the constraints normally imposed by that component (or the equivalent logical formulae) may be suspended. A diagnosis is then a *minimal* set of components for which, assuming the AB predicate true, renders the system consistent with the observations. By the word minimal, we mean that the system will be inconsistent with the observations if we assumed that AB were only true for any subset of that set of components.

More formally, we can define a diagnosis as follows:

Definition 2.1.2 (Reiter) *A* diagnosis *for* (SD, COMPONENTS, OBS) *is a minimal set* $\Delta \subseteq$ COMPONENTS *such that*

$$\text{SD} \cup \text{OBS} \cup \{\text{AB}(c) \mid c \,\epsilon\, \Delta\} \cup \{\neg\text{AB}(c) \mid c \,\epsilon\, \text{COMPONENTS} - \Delta\}$$

is consistent.

In order to compute these diagnoses, Reiter relies on two more definitions.

Definition 2.1.3 (Reiter) *A* conflict set *for* (SD, COMPONENTS, OBS) *is a set* $\{c_1 \ldots c_k\} \subseteq$ COMPONENTS *such that*

$$\text{SD} \cup \text{OBS} \cup \{\neg\text{AB}(c_1), \ldots, \neg\text{AB}(c_k)\}$$

is inconsistent.

Thus, a conflict set is a subset of components that cannot all be functioning normally at the same time, given the observations and the system description. At least one component in each conflict set must be AB. So, for the adder circuit with inputs, as in the last section, of 1, 0, and 1, and outputs of 0 and 0, a conflict occurs at the `carry-out` bit. Since the prediction of 1 for this output is computed under the assumption that AB is untrue for `x1`, `a2`, and `o1`, a conflict set for this system, with this set of observations, is {`x1, a2, o1`}.

Definition 2.1.4 (Reiter) *A* hitting set *for a collection of sets* C *is a set* $H \subseteq \bigcup_{S \epsilon C} S$ *such that* $H \cap S \neq \{\}$ *for each* $S \epsilon C$. *A hitting set for* C *is* minimal *if no proper subset is also a hitting set.*

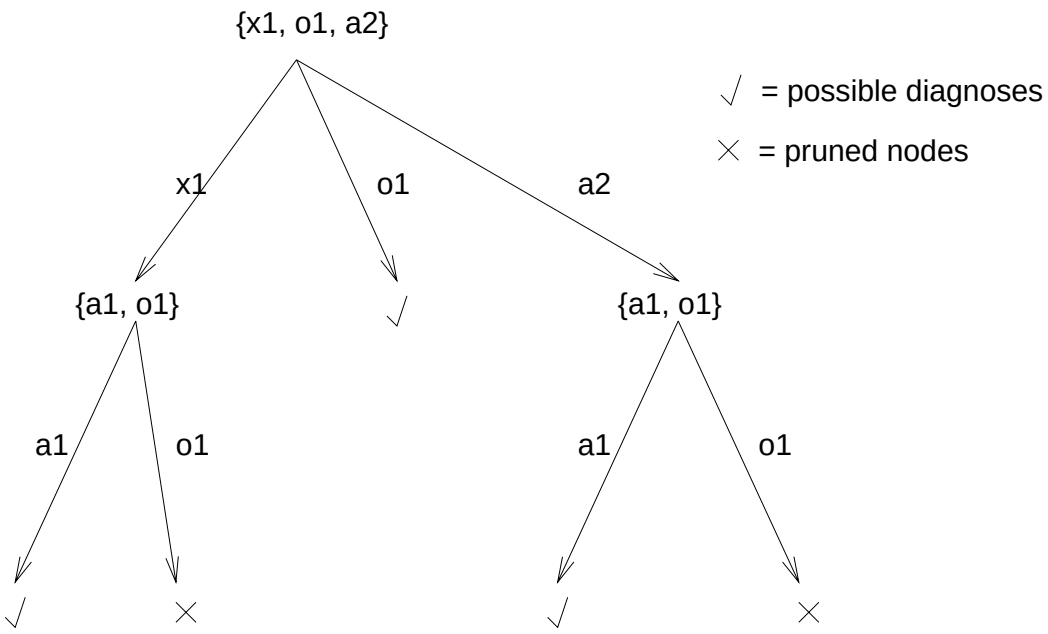

Figure 2.3: Hitting Set Tree for the Full Adder example

A hitting set is therefore a set that consists of at least one element of each of the conflict sets. Reiter proves that a diagnosis satisfying Definition 2.1.2 above is a minimal hitting set for the set of conflict sets for the system.

In the full-adder example, if we test the circuit again with all the inputs at 1, we predict values of 1 for both `bit-out` and `carry-out`. However, if we observe a value of 0 for `carry-out`, we obtain a conflict set of {`a1, o1`} since the output of `a1` is predicted to be 1, and that alone would imply an output of 1 for `o1` if it is not AB. We must thus find a minimal hitting set that hits both this conflict and the one derived above – {`x1, a2, o1`}. The set {`o1`} is one such set and is thus a diagnosis for this circuit.

These definitions suggest an algorithm for computing diagnoses. Reiter assumes that there is some theorem prover that is able to generate conflict sets. Given the set of conflict sets, his algorithm computes all the diagnoses by computing a hitting set tree (HS-tree) where each node contains a conflict set and each link is labelled with a component. An HS-tree is expanded at each leaf by labelling it with a conflict set that has not yet been hit by the set of components corresponding to the path from the root to that leaf. This enables the creation of new links corresponding to hitting each component in the conflict. When a leaf node is reached where the set of components on the path from the root to the leaf hits all the conflict sets, this set is a minimal hitting set.

Figure 2.3 is an HS-tree for the full adder example with conflict sets of {`x1, o1, a2`}

and {a1, o1}. The conflict at the root node is {x1, o1, a2} and each of these components is a label on a child link from the root node. The link labelled o1 hits both conflict sets, and thus represents a diagnosis. The other child nodes still need to hit the conflict {a1, o1}. Thus each of these has child nodes created by hitting each of the members of this conflict. However, the ones that hit o1 are pruned because the diagnoses they represent are not minimal (since {o1} is already a diagnosis). The other two do represent diagnoses. The complete list of minimal diagnoses for this set of conflicts is [{o1}, {x1, a1}, {a2, a1}].

Reiter left the computing of the conflict sets to be a problem-dependent task. A resolution theorem prover would be a general solution to the conflict set generation problem, but Reiter suggests that a more problem-specific method would be more appropriate in most cases because general theorem provers are not very efficient.

The General Diagnostic Engine (GDE) of de Kleer and Williams [1987] was an independent body of research that yielded very similar algorithms but without the same explicit logical framework. DeKleer's algorithm, like Reiter's, computes conflict sets of components that are inconsistent with the observations. However, unlike Reiter, who characterized systems as sets of formulae, deKleer characterized them as sets of constraints acting on variables. In order to compute conflicts, GDE makes assumptions about the state of components in the system and then uses constraint propagation to determine if these assumptions are consistent with the observations. An Assumption-Based Truth Maintenance system (ATMS) [de Kleer, 1986] was used to cache these computations and make conflict and test generation efficient.

An ATMS maintains *justifications* which are sets of beliefs that in conjunction, support a particular belief. These supporting beliefs may, in turn, be supported by other justifications, or they may be *assumptions*. An assumption here is a belief that supports itself. The ATMS can be used to compute *environments* for beliefs, which are underlying sets of assumptions, which, in conjunction, support these beliefs.

In its application to diagnosis, the assertions that are cached away in the ATMS are values for variables and the underlying assumptions are the components (or behavioral modes) that support the computation of that value. If some environment is a contradiction, it is said to support $\perp$ (which is the contradiction node) and is a conflict set. The ATMS has been found to be effective at saving on computational effort in GDE and similar diagnosis systems.

DeKleer's algorithm for building diagnoses is similar to Reiter's. It involves incrementally growing them by attempting to hit each conflict set. This is equivalent to Reiter's approach of building an HS-Tree. Again, the candidate diagnoses are limited to minimal sets of components.

This early work on GDE was followed by a number of extensions. DeKleer's [1991] paper is a detailed study of using probabilities of component failures for diagnosis, an idea

that was briefly introduced in [de Kleer & Williams, 1989]. In the latter paper, behavioral modes were also introduced into the GDE framework. The resulting system, SHERLOCK, is of particular interest to us since it most closely resembles the QDOCS algorithm.

SHERLOCK handles behavioral modes by having assumptions in the ATMS be assignments of *modes* to different components rather than simply whether a component is functioning correctly or not. Each mode is either a faulty or normal mode of behavior for the component. In the case of our full adder, SHERLOCK, when given the inputs and outputs from our example of the previous section, is able to come up with more detailed hypotheses, such as, for example, that the or-gate `o1` is stuck at zero and not just that it is faulty.

So, for an and-gate, the axioms would now look as follows:

$$\texttt{andg(a)} \wedge (\texttt{in1(a)} = 1) \wedge (\texttt{in2(a)} = 1) \wedge \texttt{normal(a)} \longrightarrow (\texttt{out(a)} = 1)$$

$$\texttt{andg(a)} \wedge ((\texttt{in1(a)} = 0) \vee (\texttt{in2(a)} = 0)) \wedge \texttt{normal(a)} \longrightarrow (\texttt{out(a)} = 0)$$

$$\texttt{andg(a)} \wedge \texttt{stuck-at-0(a)} \longrightarrow (\texttt{out(a)} = 0)$$

$$\texttt{andg(a)} \wedge \texttt{stuck-at-1(a)} \longrightarrow (\texttt{out(a)} = 1)$$

In other words, there would be axioms detailing an and-gate's behavior under each of its many modes. In addition to these, we must have a constraint that says that an and-gate must be in one of its three possible modes:

$$\neg\texttt{andg(a)} \vee \texttt{normal(a)} \vee \texttt{stuck-at-0(a)} \vee \texttt{stuck-at-1(a)}$$

With the inputs and outputs for this circuit as given on page 9, one of the possible diagnoses would be $\{\texttt{stuck-at-0(o1)}\}$, i.e., that the or-gate `o1`, is stuck at 0.

The many possible modes for each component increases the search space of possible diagnoses significantly.. In order to control the combinatorics, SHERLOCK uses *a priori* probabilities for each behavioral mode and a best-first search algorithm through the space of possible candidates in order to test the most likely new hypotheses first. Figure 2.4 is a slightly simplified version of SHERLOCK's algorithm from [de Kleer & Williams, 1989].[2] The predictor mentioned in step 4 of the algorithm is a constraint propagator that uses an ATMS and predicts values for other variables given observations and rules describing behavior of components under different modes. As we shall see, redesigning this step to work with dynamic systems and qualitative models is the main thrust of this thesis.

In our adder circuit example, the exact sequences of conflicts and candidate diagnoses that SHERLOCK would generate depends on the probabilities assigned to all the modes of the components. Since this algorithm forms the basis for QDOCS's algorithm, we shall examine the algorithm in more detail and consider an example later, in Chapter 3.

[2] In addition to the algorithm shown in Figure 2.4, SHERLOCK also uses Bayesian methods to predict the posterior probability of a candidate given the predictions made. Since we do not have such detailed probabilistic information for the problems to which we apply QDOCS, we omit this part of the SHERLOCK algorithm.

1. Initialize the set of consistent candidates and the set of conflicts to the empty set.
2. If there are enough candidates, stop.
3. Push the best-first search forward by finding the next highest probability candidate that hits all the minimal conflicts.
4. Focus the predictor on this candidate and use constraint propagation to determine if the candidate is consistent with the observations. If it is not, a new conflict will be generated.
5. If a conflict was generated, add it to the list of conflicts and go to step 3, else...
6. Add the current candidate to the list of consistent candidates and go to step 2.

Figure 2.4: Sherlock Algorithm

2.2 Qualitative Reasoning

The research on Qualitative Reasoning (QR) arose from a desire to capture human abilities to reason about physical systems without detailed numerical models. We are usually able to state, for example, that a bathtub that starts filling with water coming in at a steady rate will continue filling until it either reaches a point where its drain rate is equal to the inflow rate or it will reach the top and overflow. We do not need the dimensions of the tub or the exact flow rates to be able to predict that.

In order to make such predictions, a number of researchers [deKleer & Brown, 1984; Forbus, 1984; Kuipers, 1984] have proposed various methods for describing and simulating systems qualitatively. We shall concentrate on the last of these because it is the one that is most relevant to this research. The next subsection summarizes the QSIM algorithm which is the primary research product of the Kuipers group, while the subsection that follows discusses some of the achievements of the QSIM approach and summarizes some of the other approaches.

2.2.1 QSIM

A model in the QSIM framework is represented using a *qualitative differential equation* (QDE). A QDE consists of a set of variables (also known as *quantities*), their *quantity spaces*, and a set of *qualitative constraints* between them.

A quantity space for a variable consists of a sequence of *landmark values*. These are values that can be qualitatively distinguished from the rest of the quantity space. For example, in a bathtub, the landmark values for the level of water in the tub are `empty` and `full`. A level between these two values can be described as being in the interval `(empty full)`.

Qualitative constraints describe how variables relate to each other. QSIM defines a

1. Complete the initial state by solving the CSP for the given model. Put the resulting states on an *agenda*.

2. If agenda is empty, exit. Otherwise, pop a state off the agenda.

3. Generate all successor states that can immediately follow the given state by continuity.

4. Solve the CSP for the given model for these states – put all states that are consistent with the constraints of the model on the agenda. Go to step 2.

Figure 2.5: Simplified QSIM algorithm

number of possible constraint types. For example if one variable monotonically increases as another variable increases, it is described using an M+ constraint. In our bathtub example, this is the constraint that holds between the level of water and the pressure of water at the drain (which in turn is related by an M+ constraint with the outflow rate). Some of the other possible constraint types would include d/dt for derivative constraints, M– for inverse monotonic constraints, Add to describe variables which are the sum of two other variables, etc.

Qualitative constraints can also have *corresponding values*. If we know that when some particular variable is at a particular value, then other variables in the constraint are at some other given values, then we can establish a correspondence between them. For example, in the aforementioned constraint beween level and pressure, we know that when the level of the water is at `empty`, the pressure is at `0`. The M+ constraint between the two therefore has a corresponding value pair of `(empty 0)`.

Given this information in a QDE and a partially specified initial state (assignment of values to a subset of variables), QSIM produces a tree giving all the possible qualitative behaviors of the device (known as a *behavior tree*). So if we say that the bathtub has an initial state of being `empty`, and the inflow is a given constant value, QSIM will predict that the level will rise continuously until it either overflows or it reaches an equilibrium level where the outflow through the drain equals the given inflow.

A simplified version of the QSIM algorithm is shown in Figure 2.5. Since a QSIM model is simply a set of constraints, completing a partially specified state (assigning consistent values to all unspecified variables) is a constraint satisfaction problem (or CSP). QSIM completes the initial state by solving the CSP using, first, a constraint propagator, and second, a combination of the techniques of Waltz-filtering [Waltz, 1975] and backtracking. Given a set of possible states on its agenda, it picks one state and then generates successors that are continuous with it and are consistent with the constraints of the model. Continuity between two states in QSIM means that each variable in the model varies continuously

from one state to the next i.e., it does not make any discontinous jumps in magnitude or direction of change. By continuing this process of generating successor states, QSIM builds a behavior tree where each branch is a sequence of qualitative states representing a possible behavior of the system. We shall discuss the actual algorithms QSIM uses to solve the CSP later as part of our discussion of the QDOCS algorithm.

2.2.2 Alternative Approaches to Qualitative Reasoning

The QSIM algorithm has been used extensively as a research tool to perform a number of different tasks. These include modeling and simulation of various chemical plants ([Catino, 1993; Dalle Molle, 1989]), monitoring and diagnosis ([Dvorak, 1992; Lackinger & Nejdl, 1991]), compiling and simulating qualitative processes [Farquhar, 1993], and query answering in a large-scale knowledge bases [Rickel, 1995]. QSIM, however, has not been the only approach to qualitative reasoning. Some of the other approaches to QR include deKleer and Brown's Theory of Confluences [deKleer & Brown, 1984] and Forbus's Qualitative Process Theory [Forbus, 1984].

The Theory of Confluences models systems qualitatively in terms of the signs of quantities and the possible ways in which these quantities (or variables) can influence each other. There is no specific information, however, on how variables are related. For example, in the bathtub example mentioned in the previous section, the QSIM model had an M+ constraint between the level of water and the pressure at the drain. In Confluence theory, the best statement we can make about these quantities is that the sign of the level is the same as the sign of the pressure and that the sign of the derivative of level is the same as the sign of the derivative of the pressure. Unlike the M+ constraint, it does not say anything about the relationship between these quantities at any time following a given state of the system. Kuipers [1994] argues that QSIM is thus a more powerful representation for QR.

Forbus's Qualitative Process Theory (QPT) and the Qualitative Process Engine (QPE) offer another approach to the problem of QR. The difference between this approach and the QSIM approach is primarily in terms of the initial specification of the model. In QPT, the model is presented as a set of idealized *views* and *processes*. These correspond to different kinds of sets of objects, and of influencing processes respectively. For our bathtub example, we can describe a liquid container as a view and the inflow and outflow as processes that act on liquid containers. Given idealized model fragments of these objects and an initial layout of the system, the reasoning task is to construct the composite model and to perform a qualitative simulation. Farquhar [1993] has built a system called the Qualitative Process Compiler (QPC) that takes such a specification and builds a QSIM model which then gets simulated by QSIM.

QPT is thus a useful tool for model-building where there are well-defined processes and views that can be modeled conveniently but the composite model changes dynamically.

Since our research does not address the model-building problem *per se*, the models we use are built from scratch.

2.3 Qualitative Model-Based Diagnosis of Continuous Systems

In the previous sections, we have introduced some of the foundational work in Model-Based Diagnosis and Qualitative Reasoning that have influenced our research. Unfortunately, most of the work on model-based diagnosis has been in the area of static systems such as combinational circuits, and not the continuous dynamic systems of interest to us.

In this section we will look at some of the few exceptions to this that our research is built upon. In the first subsection we will look at the concept of monitoring in general and specifically at the MIMIC system [Dvorak, 1992], while in the second subsection we will look at INC-DIAGNOSE [Ng, 1991]. In the final subsection we will motivate our own approach.

2.3.1 Monitoring and MIMIC

The problem of diagnosis as applied to continuous dynamic systems needs to be looked at in the larger context of *monitoring* systems. A monitoring system is generally defined in the literature ([Dvorak, 1992], [Doyle, Sellers, & Atkinson, 1989] and others) as an on-line system that receives numerical values from sensors placed appropriately on a device and tracks the behavior of the device in real-time. Traditionally, a monitoring system will use the information from the sensors as input to a program that sets off alarms when the device is behaving abnormally. These systems, which are currently installed in most modern industrial plants, usually trigger alarms only when certain sensor inputs fall above or below predetermined threshold levels.

Modern monitoring systems as envisioned by AI researchers (e.g.,[Dvorak, 1992; Doyle et al., 1989; Lackinger & Nejdl, 1991]), go beyond this and try to simulate a model of the device in order to predict its behaviour and to use these predicted values to detect faults. This is a more sophisticated technique than the traditional threshold alarm and is a necessity if the monitoring system is for devices that do not normally operate in equilibrium.

The relationship between monitoring and diagnosis is as follows: the monitoring system detects that there are faults, while the diagnosis system isolates them. Both MIMIC [Dvorak, 1992] and DIAMON [Lackinger & Nejdl, 1991], consider the diagnostic system to be an essentially distinct adjunct to the monitoring system.

Our research focusses exclusively on the problem of diagnosis in such systems. There are a number of issues within the monitoring domain which are important research problems and have partial solutions in the literature, but which we shall not address as part of this

research. Among these are the issues of how often to sample the sensors in order to ensure that the monitoring system does not miss any qualitatively significant states; how the numerical values obtained from the sensors are mapped to landmarks from the qualitative model; how discrepancies between the predicted and sensed values are detected in the face of the ambiguity of the quantitative-qualitative mapping, etc.

For the purposes of this research, we assume that some monitoring system is tracking the behavior of the device using qualitative models, and when a discrepancy is detected, it takes a sequence of qualitative sensor readings, and passes it along to the diagnosis system. Thus the input to the QDOCS system is simply a sequence of qualitative values from a given set of sensors. The output is one or more diagnoses.

MIMIC uses a technique known as *dependency tracing* in its diagnostic module. Since MIMIC set the standard as the main model-based diagnostic system to come out of the QSIM research project, we think it is worthwhile to explain this technique and our motivation for moving away from the dependency tracing algorithm.

In addition to a QSIM model of the device where constraints are linked with components, MIMIC requires a model of the component structure of the device with information on which components can influence which other components. For example, if we have a tank with an outlet through which water flows out into another tank (the classic cascaded tanks example that is used in [Dvorak, 1992]), it is impossible for a fault, such as a leak or a clogged drain, in the tank below, to have any effect on the tank above. This information is encoded in a *dependency graph*. MIMIC uses this dependency graph to suggest candidate hypotheses in the following manner: given that a particular sensor shows a value that is inconsistent with the QSIM predictions of the model of the device, the dependency tracer looks upstream in the dependency graph of possible components that could have an effect on the reading of the sensor. All fault modes of all these components are considered as possible candidates for faults. However, for efficiency reasons, MIMIC makes a single-fault assumption.

This technique is a fairly efficient one for producing hypotheses given the right influence model. However, it suffers from a number of drawbacks which we shall list here.

First, it requires the model-builder to come up with a whole separate component-connection model for the device. Such models are not obvious and may take some effort to generate. For example, consider a model for two tanks connected through a pipe. One may be tempted to assert that faults in the pipe only affect the tank that the pipe drains into. However, if the pipe becomes clogged or if the fluid backs up through the pipe, it may also affect the amount of water in the draining tank. Thus the effects of components have to be considered under all possible fault scenarios before one can assert the directionality of the effects of a component.

The second drawback to the dependency tracing algorithm is that in large devices

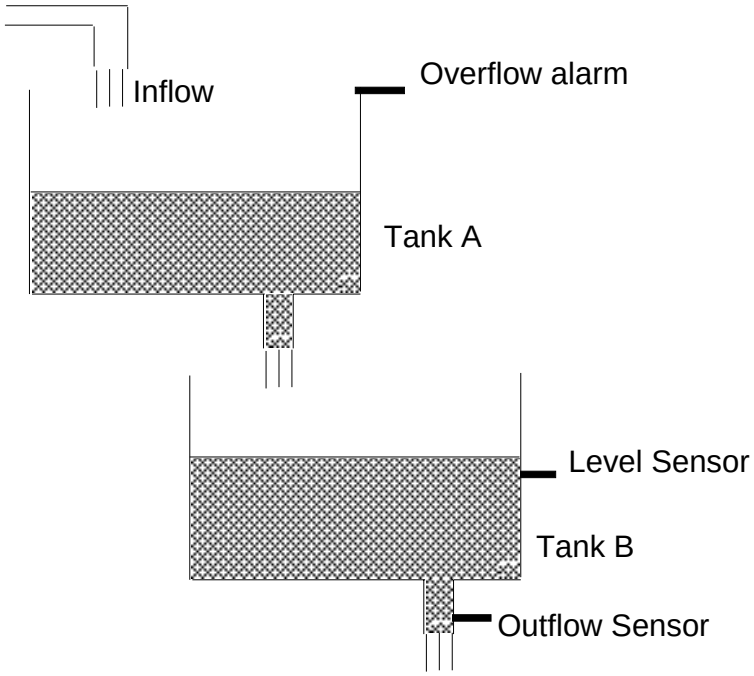

Figure 2.6: Cascaded Tanks

with many complex parts there are just too many dependencies going in all directions. Neat divisions in the dependency graph, as in the cascaded tanks, are rare. This means that most anomalous behaviors will result in a large number of possible components that could have been responsible, and a large number of candidate hypotheses will have to be considered.

The third, and perhaps most important, reason for abandoning the dependency tracing approach of MIMIC is that a given fault may not even be causally upstream from the sensor that gave the anomalous reading. A simple example will illustrate this. Consider the structure of the cascaded tanks shown in Figure 2.6. The model has a break in the causal dependency graph since nothing in Tank B can affect anything in Tank A.

We assume that there is a level (or amount) sensor and an outflow sensor in Tank B, but only an overflow sensor in Tank A. Now suppose we start simulating the model and match sensor readings to qualitative states as MIMIC does. Furthermore, suppose we gauge over a certain period of time that the level of Tank B is falling but that the flow rate out of Tank B is apparently not high enough to account for the flow. Given some information in the model on the relationship between level and flow rates in Tank B, we may be able to deduce that the level in Tank A must be falling.

Now suppose that the overflow alarm in Tank A is triggered. This contradicts our prediction that the level in Tank A was falling. Given this information, MIMIC would use

causal dependency tracing to look upstream in the dependency graph from the sensor where the contradiction was detected to find all possible single faults.

Since this fault was located at the overflow sensor, the only places where MIMIC might detect faults are within Tank A and in the input stream. This ignores the possibility that there are faults in one or both of the sensors in Tank B which were responsible for our computation of a falling value for the level in Tank A.

The basic problem with this mode of reasoning is that if predictions are to be made in acausal directions, then the predictions are dependent on components that are not causally upstream from them. Since QSIM propagates values without regard to causality, dependencies must be propagated in all directions, not simply causally upstream.

If these causal dependencies are not used, MIMIC's diagnosis algorithm simply becomes a generate-and-test algorithm. A generate-and-test algorithm would simply generate faults in a predetermined order and test them by attempting to simulate the given behavior. The actual order in which faults are generated can vary – in a domain where probabilities of faults are known, the most reasonable order would be a most-probable first ordering. Such an ordering would allow the generator to produce single as well as multiple faults. We use such a generate-and-test algorithm as a baseline system to compare QDOCS against in our experiments in Chapter 5.

2.3.2 INC-DIAGNOSE

The QDOCS algorithm is based on some of the ideas proposed by Ng [1990, 1991]. Ng's algorithm INC-DIAGNOSE was an early attempt at solving the same problem as QDOCS and is modeled after Reiter's algorithm which we introduced in Section 2.1.3.

Ng extended Reiter's algorithm by using QSIM's constraint propagator in place of the theorem prover that Reiter assumes. It incrementally uses constraint propagation on each of the input qualitative states to build conflict sets and then uses these conflicts to build a Hitting Set Tree (HS-Tree) just as in Reiter's algorithm. In order to explain this algorithm, we will need to go into some detail into QSIM's constraint propagation algorithm.

As we discussed in the previous chapter, QSIM must solve a constraint satisfaction problem (CSP) as its first step towards producing a qualitative simulation. QSIM combines a variety of approaches to this problem. The three main steps of its algorithm are:

1. Constraint Propagation,
2. Waltz-filtering, and
3. Backtracking.

We shall look at the Waltz-filtering and Backtracking algorithms in the next chapter as they form important parts of the QDOCS algorithm. QSIM's Constraint Propagation is an

Variable	Values
A	$0\ a_0\ a_1$
B	$0\ b_0\ b_1$
C	$0\ c_0\ c_1$
D	$0\ d_0\ d_1$
E	$0\ e_0\ e_1\ e_2$

Constraint	Corresponding Values
(M+ A B)	$(0\ 0)\ \ (a_0\ b_0)\ \ (a_1\ b_1)$
(Mult B C E)	$(0\ 0\ 0)\ \ (0\ c_0\ 0)\ \ \ldots\ \ (b_0\ c_0\ e_1)\ \ \ldots$
(Add C D E)	$(0\ 0\ 0)\ \ (0\ d_0\ e_0)\ \ (c_0\ 0\ e_0)\ \ (c_0\ d_0\ e_1)\ \ \ldots$
...	...

Figure 2.7: A simple QSIM Model

important part of both INC-DIAGNOSE and QDOCS and hence we shall discuss it here.

Figure 2.7 is a representation of a simplified QSIM model we will use to illustrate QSIM's constraint propagation algorithm. The first of these constraints, (M+ A B), is one where the value of either variable is completely determined by the value of the other. So, for example, if we know that A has value a_0, a simple constraint propagator can conclude that B must have the value b_0. As we shall see, however, it will not always be possible to determine a value for a variable given values for other variables in the constraint network using propagation alone. In such cases, we say that constraint propagation is *blocked*.

Simple propagation is obviously a very efficient way of solving the CSP for a given network. The algorithm is $O(n^2)$ in the number of constraints in the network. INC-DIAGNOSE, like most other algorithms in the literature [de Kleer & Williams, 1987; Lackinger & Nejdl, 1991] is based on the assumption that propagation through the constraint network will always be possible. Given a QSIM model and a set of initial values for some variables, INC-DIAGNOSE propagates the effects of these initial values to complete the state while checking consistency with each of the constraints. While propagating values, the system also keeps track of which constraints are responsible for producing which values. Since these constraints are due to particular components of the device, this is equivalent to tracking the effects of the components of the device. If any of the constraints is inconsistent with the values of the variables it acts upon, INC-DIAGNOSE returns, as a conflict, the set of

all the components responsible for these values. INC-DIAGNOSE computes conflict sets for different qualitative states of the given behavior and uses these conflicts as input to Reiter's algorithm to compute all the possible minimal diagnoses.

Unfortunately, a general constraint network will not always allow simple propagation. We can easily see this in the constraint model shown in Figure 2.7. Suppose we know that `B` has the value b_0 and `D` has the value d_0. Consider the constraints `(Mult B C E)` (i.e., that `B` $\times$ `C` $=$ `E`) and `(Add C D E)` (or `C` $+$ `D` $=$ `E`). An analysis of the corresponding values listed for these constraints would reveal that the only consistent values for `C` and `E`, given the above values for `B` and `D`, are c_0 and e_1 respectively. However, when we attempt to propagate values through these constraints we discover that since we only have values for one variable of each of these constraints, we cannot get any more information from propagation.

The fact that propagation here failed to find values for these variables does not imply that the network is consistent. Suppose we had yet another constraint `(M+ C E)` with corresponding values including the pair (c_0 e_0). Since in our example the only values for `C` and `E` are c_0 and e_1 respectively, this constraint causes the state to be contradictory. However, with this new set of constraints, we would still not be able to get values for `C` or `E`, and thus would not be able to detect the contradiction.

This incompleteness of INC-DIAGNOSE's propagation algorithm, due to its reliance on QSIM's propagator as its only means of generating conflicts, is a major limitation on its diagnostic capability. The effect of using an incomplete propagation procedure to compute conflict sets is that INC-DIAGNOSE may produce unsound diagnoses. This is because when a contradictory hypothesis is tested, it may well be found consistent and thus presented as a diagnosis even though it contradicts the observations.

2.3.3 Motivation for QDOCS

The goal of our research has been to attempt to correct the flaws associated with the precursor algorithms we have discussed in the previous sections. We do this in a number of ways.

First, because QSIM uses a general constraint-based language and has been proven to be a useful tool in describing and simulating dynamic systems qualitatively, it is the ideal language for a general qualitative diagnostic system.

Second, QSIM produces a disjunction of possible behaviors for any given model. This means that while we may be able to show that a particular sequence of observations is consistent with a particular model, we cannot show that the model logically implies the observations as we would need to in an abductive system. This is why, a QSIM-based diagnostic system would have to use a consistency-based diagnosis algorithm, rather than an abductive one.

Third, since components of dynamic systems often fail in predictable ways, a diag-

nosis algorithm for such systems should have the ability to make use of behavioral-mode information in doing diagnosis.

These considerations all prompted us to choose to build a SHERLOCK-based diagnosis system with QSIM providing a constraint language. Furthermore, we wanted to avoid the specific problems with acausal predictions that MIMIC exhibits and the incompleteness problem of INC-DIAGNOSE. We avoid the acausal prediction problem by using dependencies computed from the actual prediction process rather than a post-prediction causal analysis. The incompleteness problem of INC-DIAGNOSE is overcome by incorporating more general constraint satisfaction and across-time verification algorithms adapted from QSIM, and incorporating dependency propagation into these procedures.

Chapter 3

Basic Diagnosis Algorithm

This chapter presents the knowledge representation and algorithms that QDOCS uses to generate diagnoses from a sequence of qualitative states representing a behavior that is inconsistent with the normal model of a given system. There are many distinct procedures in QDOCS which we look at in turn. The first section discusses the knowledge representation that QDOCS uses, while the remaining sections all look at parts of the QDOCS algorithm.

3.1 Representing Knowledge

Most of the information that QDOCS needs comes from the QSIM model of the system. However, there is some information that QSIM models do not provide that is essential to the diagnosis task. In order to illustrate the knowledge representation we use and our algorithm, we introduce a simple diagnosis problem as an example.

Consider the bathtub shown in in Figure 3.1 It is assumed that this bathtub is monitored by sensors measuring the amount of water in the tub and the flow rate of the water through the drain. Some of the faults that can be posited about this system include a blocked drain, and sensors stuck at various levels. The bathtub is first represented using a QSIM model as shown in Figure 3.2. QSIM allows the model builder to define discrete variables that can be used to define conditions on constraints and thus allows the combination of distinct models into one description. The description is thus a constraint model with the mode variables acting as parameters that turn constraints on and off depending on the active faults of the system. So, in the parameterized constraints for the drain shown in Figure 3.2, if the variable `drain-mode` has the value `blocked`, the model specifies that the `outflow` must have value zero and be steady. If, on the other hand, the drain is in a `normal` mode, the outflow is related to the pressure through an M+ (monotonically increasing) constraint. Finally, we also allow the drain to be in an `unknown` mode where the quantity that the drain controls, i.e., the `outflow` remains unconstrained. This facility is included to

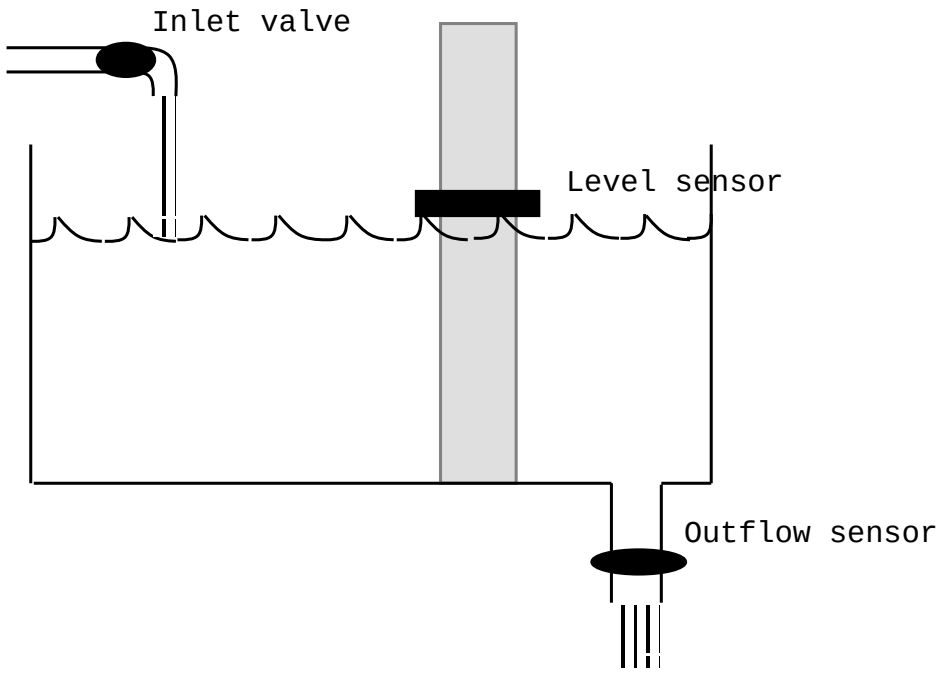

Figure 3.1: The Bathtub Example

allow for unanticipated faults in the device.

In general, the conditions for activating a constraint may be any logical expression built using conditions on mode variables (e.g., `(and (comp1-mode mode1) (comp2-mode mode2))`. We shall refer to these expressions as *mode expressions*.

In order to do diagnosis, we will also need to provide a method for associating the mode variables of the QSIM model with components. To achieve this, an additional structure must be provided. QDOCS provides the *defComponents* facility for this purpose. A defComponents description consists of a number of clauses corresponding to each of the components in the system. The defComponents description for the bathtub example is shown in Figure 3.3. The first element of each clause is the name of the component (e.g., `drain`). The second element is the mode variable in the QSIM model associated with it, e.g., `drain-mode`. The remaining elements are the actual values that these mode variables can have as well as their *a priori* probabilities. These probabilities are used by QDOCS to rank candidate diagnoses and select the most probable ones. We make the simplifying assumption that the components are independent. This means that the *a priori* probability of two components being in two given modes is simply the product of their individual mode probabilities.

The input to QDOCS consists of one or more consecutive qualitative states of the

```
Quantity Spaces
    (amount    (0 FULL))
    (level-sensed (0 FULL))
    (level     (0 FULL inf))
    (pressure  (0 inf))
    (outflow   (0 inf))
    (outflow-sensed (0 inf))
    (inflow    (0 if* inf))
    (netflow   (minf 0 inf))
Discrete Variables
    (flowsensor-mode (normal stuck-at-0 unknown))
    (levelsensor-mode (normal stuck-at-0 stuck-at-full unknown))
    (inletvalve-mode (normal stuck-closed unknown))
    (drain-mode (normal blocked unknown))
Constraints
    ((M+ amount level)              (0 0) (full full))
    ((M+ level pressure)            (0 0) (inf inf))
    (mode (drain-mode normal)
          ((M+ pressure outflow)          (0 0) (inf inf)))
    (mode (drain-mode blocked)
          ((zero-std outflow)))
    (mode (flowsensor-mode normal)
          ((equal outflow outflow-sensed) (0 0) (inf inf)))
    (mode (flowsensor-mode stuck-at-0)
          ((zero-std outflow-sensed)))
    (mode (levelsensor-mode normal)
          ((equal level level-sensed) (0 0) (full full)))
    (mode (levelsensor-mode stuck-at-0)
          ((zero-std level-sensed)))
    (mode (levelsensor-mode stuck-at-full)
          ((constant level-sensed full)))
    (mode (inletvalve-mode normal)
          ((constant inflow if*)))
    (mode (inletvalve-mode stuck-closed)
          ((zero-std inflow)))
    ((ADD netflow outflow inflow))
    ((d/dt amount netflow))
```

Figure 3.2: The Combined Bathtub Model

```
(defComponents bathtub
   (drain drain-mode (normal 0.89) (blocked 0.1) (unknown 0.01))
   (levelsensor levelsensor-mode (normal 0.79) (stuck-at-0 0.1)
                (stuck-at-full 0.1) (unknown 0.01))
   (flowsensor flowsensor-mode (normal 0.89) (stuck-at-0 0.1)
               (unknown 0.01))
   (inletvalve inletvalve-mode (normal 0.89) (stuck-closed 0.1)
               (unknown 0.01)))
```

Figure 3.3: The Bathtub Component Structure

sensors of the device. In the case of the bathtub, these would be the sensed versions of the level and the outflow. A typical pair of qualitative states might be:

State 1: `(level-sensed (0 full)) (outflow-sensed 0)`
State 2: `(level-sensed full) (outflow-sensed 0)`

This means that in the first observation, the level is sensed at somewhere between the landmarks `0` and `full` and the outflow is sensed at `0`, while in the next observation, the level sensed is `full` while the outflow sensed is still `0`. The output of QDOCS is a set of assignments to all the mode variables. For example, one of the diagnoses for the above observations is

`(drain-mode blocked) (levelsensor-mode normal) (flowsensor-mode normal)`
`(inletvalve-mode normal)`

In other words, the drain is blocked and the other three components are behaving normally. We will sometimes omit `normal` components while describing diagnoses. Thus the above hypothesis would be `(drain-mode blocked)`.

In the following sections we shall look at how such diagnoses are computed. We begin with a discussion of the general strategy that QDOCS uses and follow it with sections on the component parts of the algorithm.

3.2 General Strategy

The diagnostic approach of QDOCS is similar to that of SHERLOCK [de Kleer & Williams, 1989] as introduced in the previous chapter. Like SHERLOCK, QDOCS uses a best-first search mechanism and focusses its search on the leading candidate diagnoses as determined by their *a priori* probabilities. Figure 3.4 lists the steps of the algorithm. QDOCS maintains an *agenda* of hypotheses to be tested and a list of conflict sets. The former is initialized to

Diagnose (BEH)

1. Initially, set **agenda** to the singleton set consisting of the hypothesis that all components are behaving normally. Set **conflicts** to the empty set.

2. Choose the diagnosis, D, on the agenda that has the highest *a priori* probability.

3. If there is a conflict C in **conflicts** that is not *hit* by D, then expand **agenda** by considering all hypotheses generated by changing the mode value of a single component from its value in D in order to hit C, and only keeping those that hit all the conflict sets generated thus far. Go to Step 2.

4. Test the hypothesis D by calling check-hypothesis with BEH as the behavior to be checked. Let C be the return value of check-hypothesis.

   - If C is nil, then D is a viable hypothesis that would explain the given behaviour. Add D to the list of final hypotheses. If we have enough hypotheses, then return this list and exit.
   - C is a new conflict set. Add it to the set **conflicts** and expand **agenda** by considering all hypotheses generated by changing the mode value of a single component from its value in D in order to hit C, and only keeping those that hit all the conflict sets generated thus far. Go to Step 2.

Figure 3.4: QDOCS's Top Level Algorithm

the single hypothesis that everything is functioning normally while the latter is initialized to the null set.

The hypothesis checker, which is described in the next section, is first called with the initial hypothesis of all the components being normal. If it returns a null value, the given behavior is consistent with the hypothesis; in other words, the given behavior is a possible result of running QSIM on the model assuming all component mode variables are in the `normal` mode. If there is no QSIM simulation that results in the given behavior, the checker returns a conflict set of component mode variable values. This conflict set is then added to the set of conflict sets, and the agenda is expanded by adding all hypotheses generated by changing the mode value of a single component in such a way that it *hits* all the conflict sets. An example will help illustrate this process.

Suppose we are given the behavior

State 1: `(level-sensed (0 full)) (outflow-sensed 0)`
State 2: `(level-sensed full) (outflow-sensed 0)`

for our bathtub example of Figure 3.1. QDOCS will initially query the hypothesis checker with the hypothesis that all component mode variables are in the `normal` mode. Since there is no QSIM simulation corresponding to this behavior under that hypothesis, QDOCS will return some conflict. If the conflict returned is, say, {`(drain-mode normal), (levelsensor-mode normal), (flowsensor-mode normal)`},
what is known about the system is that the drain, the level sensor and the flow sensor, cannot all be normal at the same time. This means that the possible hypotheses that could be tested now include

- that the drain is blocked,
- the drain is in some unknown state,
- the level sensor is stuck at zero,
- the flowsensor is stuck high,
- and so on...

Each of these is a slight perturbation of the initial normal hypothesis that accounts for the conflict set. These hypotheses are ranked in order of decreasing *a priori* probability and they are placed on the agenda. QDOCS then chooses the hypothesis on this agenda with the highest probability and repeats the process with this as the new current hypothesis.

At each point, when a new hypothesis is chosen from the agenda, it needs to be checked against the set of existing conflicts to ensure that no hypothesis is tested that does not hit all the known conflict sets. This is because a conflict set may have been detected

since the time when the hypothesis was placed on the agenda. If a conflict set is found that is not hit by the hypothesis, the agenda is immediately expanded with hypotheses generated by changing the mode value of a single component in such a way that it hits all the conflict sets. If no such conflict is found, the hypothesis checker is called and the resulting conflict set is used to expand the agenda.

This process continues until one or more hypotheses is found that is consistent with the given behaviour. We may choose to look at some n number of possible explanatory hypotheses or limit ourselves to one.

We can state and prove a theorem that this procedure is always guaranteed to compute the most probable hypothesis that is consistent with the given observations. The proof relies on the fact that we start with the most probable hypothesis (all components behaving normally) and every component faulted in any way reduces the probability of the resulting hypothesis. In other words, the result holds in situations where the `normal` mode of a component is always more probable than any faulty mode. It also relies on the correctness of the **Check-Hypothesis** procedure. Correctness for this procedure means that if **Check-Hypothesis** is called with a hypothesis for which the given observations are a possible behavior, then it signals this to **Diagnose**, otherwise it returns a correct conflict set, i.e., a set of component mode values whose conjunction is true under the hypothesis but is inconsistent with the given observations.

The proof of the theorem will be analogous to the proof of the admissibility of the A* algorithm given by, among others, Nilsson [1980]. However, when we look at the space of hypotheses that **Diagnose** considers as a graph search problem, we do not know immediately that the most probable consistent hypothesis is actually in the graph. This is because the successors generated by this algorithm are dependent on the conflicts returned by **Check-Hypothesis** and while there are constraints on what this function can return, there is still a range of possibilities.

Theorem 3.2.1 *If the* `normal` *mode of every component has the highest* a priori *probability of all the modes of that component, assuming all the components fail independently, and assuming the correctness of the* **Check-Hypothesis** *procedure, the* **Diagnose** *procedure, given a sequence of observations, will produce the most probable hypothesis that is consistent with the observations.*

Proof: Let D be the most probable hypothesis consistent with the observations. D consists of an assignment of mode values to all the components of the device. Of these, let F_D be the set of all mode assignments in D that are not `normal`. Finally, let I be the initial hypothesis that says that all the components are `normal`.

Let H be any hypothesis which has $F_H \subseteq F_D$, as its only faulty mode assignments (i.e., all the other components are `normal`). Note that I is one such hypothesis. Because of

the independence of components and the higher probability of `normal`, the overall probability of H must be higher than that of D. Given this, and our assumption that D is the most probable hypothesis that is consistent with the observations, H must be inconsistent with the observations.

If H is the most probable hypothesis on the agenda, the **Diagnose** algorithm will find some conflict C that is not hit by H, either by calling **Check-Hypothesis** or by looking in the Conflicts list if such a conflict already exists. Since D is consistent with the observations and thus hits all the conflict sets, it must also hit C. If D hits C and H does not, there must be some component mode value in $D - H$ that is different from its value in C. However, the only differences between D and H are in the set of components that are `normal` in H, but faulty in D. Thus C must contain the `normal` mode value for some such component (call it c).

In the agenda-expanding steps of **Diagnose**, the procedure always generates all hypotheses that hit the conflict set by changing the mode value of one component from its value in C to any different value. Thus one new hypothesis that will be generated will be the hypothesis derived from H by changing the mode value of component c from `normal` to its value in D.

We have thus shown that when any hypothesis H is considered whose set of faults is a subset of the set of faults in D, one of the hypotheses that will be generated and placed on the agenda will be one whose set of faults simply adds another of the faults in D. Therefore, by induction, starting with the hypothesis I, we can see that expanding the right hypotheses will always result in a sequence of hypotheses that will terminate in D.

It must also be true that at any time before hypothesis D is tested, some hypothesis that is on this sequence from I to D must be on the agenda. This follows from the fact that I is on the path and is on the initial agenda. and the expansion of any hypothesis on this sequence produces another hypothesis on the sequence.

The remaining work is to prove that **Diagnose** will generate such a sequence of hypotheses and test the hypothesis D before testing any other hypothesis consistent with the observations. Suppose that there is some consistent hypothesis $D*$ which is tested before D. By our initial assumption, D has a higher probability than $D*$. Since we showed that all of the states on the sequence from I to D inclusive have higher probability than D, it follows that all of these hypotheses have higher probability than $D*$. This means that whenever $D*$ is expanded, some hypothesis in the sequence from I to D will be on the agenda and have higher probability than $D*$. Since **Diagnose** always chooses the hypothesis on the agenda with the highest probability to expand, this contradicts our premise that $D*$ is expanded before D.

□

The part of the algorithm that we have not yet discussed and this proof is dependent

on, is the hypothesis checker. In the following section, we will discuss this module and see how QDOCS computes a conflict set given a hypothesis and a behavior.

3.3 The Hypothesis Checker

The approach that QDOCS takes to check hypotheses and to generate conflict sets is to attempt to perform a QSIM simulation that matches the given qualitative behavior. By keeping track of dependencies during this process, QDOCS is able to generate conflict sets if it fails to produce the observed behavior.

The hypothesis checker can again be broken up into two distinct phases corresponding to the parts of the QSIM algorithm. In the first phase, we attempt to complete states for each of the given qualitative states to check if the set of sensor values itself is consistent with the given hypothesis. If any of the states is inconsistent, the conflict set generated by attempting to complete the state is returned. If all the states are consistent by themselves, the hypothesis checker enters the second phase where it must attempt a simulation across time to check if a continuous chain of states can be made corresponding to the given behavior. Such a chain of states would constitute a QSIM simulation of the behavior.

We will look at each of these phases in turn in separate subsections. In particular, since the first phase is actually carried out with two separate procedures, we look at these separately and the third part discusses the second phase of the hypothesis checker.

A key idea that will be used throughout the discussion of the hypothesis checker is the association of *dependencies* with variables. The following definitions will clarify the meaning of this term as we will use it here.

First, we need to clarify what it means for a component to be associated with a constraint.

Definition 3.3.1 *A component* Comp *is* associated *with a constraint* Con *in the model* M *if the mode variable corresponding to* Comp *in the defComponents definition for* M *appears somewhere in the mode expression for* Con.

Note that since mode expressions may be arbitrary logical formulae built with conditions on mode variables, there may be many components associated with a given constraint.

During the constraint satisfaction and propagation process, constraints have *possible tuples* associated with them. These are sets of assignments to each of the variables the constraint acts upon that are considered possible given the application of the constraints applied thus far. Variables have *possible values* associated with them that are, similarly, the values considered possible for that variable given the application of constraints tried thus far. During the propagation process, these values are implicit because only single values for qualitative magnitudes and directions are associated with each variable, where it is possible

to compute such values. In both cases, however, we will define *dependency sets* as the set of mode variable assignments to components that were used to determine the computed values.

Dependencies are defined recursively as follows:

Definition 3.3.2 *When no constraints have yet been applied on the constraint net, all constraints have empty dependency sets. If the application of constraint* Con *reduces its set of possible tuples,* Con *is said to be* dependent *on the mode values of the components associated with it, and these mode values must be in its dependency set. If the number of possible tuples for* Con *is reduced because one of the variables it acts upon,* var *has had its set of possible values reduced,* Con *is* dependent *on all the component mode values that* var *is dependent on, and all of these mode values must be in* Con*'s dependency set.*

Similarly, for variables,

Definition 3.3.3 *When no constraints have yet been applied on the constraint net, all variables have empty dependency sets. If a reduction of the set of possible tuples for a constraint* Con *reduces the set of possible values of a variable* var *it acts upon,* var *is said to be* dependent *on all the component mode values that* Con *is dependent on, and these mode values must be in its dependency set.*

Given these definitions, it is easy to see that the following is true.

Lemma 3.3.1 *During a constraint satisfaction or propagation process, if a variable V has possible values $\{v_1, \ldots, v_n\}$ and dependencies $\{A_1, \ldots, A_m\}$, then*

$$T \wedge A_1 \wedge \ldots \wedge A_m \longrightarrow (V = v_1) \vee \ldots \vee (V = v_n)$$

where T represents the model of the system including the initial values specified in the constraint satisfaction or propagation process.

Proof: The proof follows from a simple induction on the number of reduction actions carried out over the constraint network. The base case is after zero applications and follows trivially from the definitions of dependencies above. In the inductive case, if the assertion is true after n reduction actions, the *(n+1)*th action will either reduce the number of possible values for a variable due to a constraint or the number of possible tuples for a constraint, because of a variable. In each case, the definitions of dependencies above ensure that the condition will hold after the *(n+1)*th action. We shall show this in one case; the other kinds of constraint-based reductions have analogous proofs.

Suppose the *(n+1)*th action is one where V, which previously had $\{v_1, \ldots, v_n\}$ as its set of possible values, loses values $v_{i+1}, \ldots, v_n$ because of constraint C that acts upon V. Suppose also that the V, prior to this application of C, has dependencies $\{A_1, \ldots, A_m\}$,

while C has a dependency set of $\{B_1, \ldots, B_k\}$. According to Definition 3.3.3, the dependency set of V must now be updated to be $\{A_1, \ldots, A_m, B_1, \ldots, B_k\}$. By our inductive assumption,

$$T \wedge A_1 \wedge \ldots \wedge A_m \longrightarrow (V = v_1) \vee \ldots \vee (V = v_n)$$

Since the constraint C and its remaining tuples are inconsistent with $(V = v_{i+1}) \vee \ldots \vee (V = v_n)$, we have

$$T \wedge B_1 \wedge \ldots \wedge B_k \longrightarrow \neg((V = v_{i+1}) \vee \ldots \vee (V = v_n))$$

Resolving these we get

$$T \wedge A_1 \wedge \ldots \wedge A_m \wedge B_1 \wedge \ldots \wedge B_k \longrightarrow (V = v_1) \vee \ldots \vee (V = v_i)$$

Since the dependency set of V according to Definition 3.3.3 must now be updated to be $\{A_1, \ldots, A_m, B_1, \ldots, B_k\}$ and V's possible values are now $\{v_1, \ldots, v_i\}$, and since nothing else in the constraint network has changed, the above equation proves our inductive hypothesis.

□

This result leads us to the following conclusion.

Theorem 3.3.1 *When some variable* var *has no possible values, its dependency set is a conflict set. Alternatively, if constraint* con *has no possible tuples, its dependency set is a conflict set.*

Proof: Considering each part of the theorem separately,

- Since each variable in a QSIM model corresponds to a physical quantity in the device, it must have a value in every state. If our application of the constraints of the model reveals that the variable cannot have a value in a particular state, and its dependency set is $\{A_1, \ldots, A_m\}$, then by Lemma 3.3.1, the conjunction of these assumptions implies the null disjunction (or empty clause) which, by definition, is a contradiction. Hence $\{A_1, \ldots, A_m\}$ is a conflict.
- If a constraint has no possible tuples, each of the variables it acts upon has no possible values. Thus the same argument as above holds.

□

This theorem points to the approach we use throughout the constraint propagation and satisfaction algorithms in QDOCS to yield conflict sets. Dependency sets are always maintained for all variables and constraints in the model, and whenever a variable has no possible values, or a constraint has no possible tuples, its dependencies are returned immediately as a conflict set.

3.3.1 Constraint Propagation

As mentioned in the previous section, the first phase of the hypothesis checker involves checking if each individual set of sensor readings is consistent with the hypothesis. The first part of this phase is very similar to the constraint propagation algorithm of INC-DIAGNOSE and is based on QSIM's constraint propagation algorithm. For the reasons outlined in Chapter 2 this algorithm cannot always find inconsistencies between a given set of sensor values and the model when they exist. However, the algorithm is still useful because it is more efficient than the constraint satisfaction algorithm we will outline in the next section and can be used to reduce work during the constraint satisfaction phase.

Figure 3.5 is a listing of the QSIM Propagation algorithm updated to maintain dependencies. Each constraint is first checked for consistency with the known values of variables it acts upon and if it is consistent, any information that can be directly concluded regarding the qualitative magnitude or direction of change of the variable, is asserted. Whenever a variable is changed all other constraints acting on it are again considered for propagation. This continues until all the constraints have been considered for propagation and there are no more new values for variables which could cause others to change.

Dependency maintenance occurs simultaneously with constraint propagation. We follow the definitions given in the previous section - whenever a constraint causes some variables to change, the dependency set on the constraint is updated to include the dependencies of all the variables that have values, and the dependencies of the changed values are updated to include the dependencies of the constraint. Whenever some constraint reveals a contradiction, its dependencies are returned as a conflict.

If we run out of active constraints, the propagator simply exits. However, the values of the variables computed as well as the dependency sets are preserved for use in the constraint satisfaction algorithm discussed in the next section.

To illustrate this algorithm, consider the constraint network for the bathtub model shown in Figure 3.6. These constraints are the ones that are active if all the components are assumed to be normal. Figure 3.6 also shows the qualitative magnitudes (qmags) for the first set of sensor values from the example introduced in Section 3.2. Initially, since the qmags of the sensed variables are known, these are initialized to their respective values. The propagation algorithm of Figure 3.5 initially chooses a constraint and attempts to propagate the values.

Most of the possible initial choices for constraints in the network will actually have no effect since most of the variables do not have initial values. However, when the constraint `(EQUAL Level Levelsensed)` is chosen, the propagation function for `EQUAL` is called which sets Level to the interval `(0 full)`. This causes the constraints `(M+ Amount Level)` and `(M+ Level Pressure)` to get put back into the variable **Constraints** (unless they are already members of this list) since there is new information in the variable `Level` that can be propagated. It

Propagate

1. Initialize stack of active constraints (**constraints**) to all the constraints in the model. Initialize the magnitudes and directions of all unknown variables to `NIL` while initializing values of known variables as given.

2. Pop **constraints** to get **con**.

3. Depending on the type of **con**, call a function to check if the values of the variables that **con** acts upon are consistent with **con**. If they are inconsistent, exit immediately, returning the dependency set of **con** as a conflict.

4. Depending on the type of **con**, call a function to propagate values across **con**. These functions will assert as much as is possible to assert about the values of the unknown variables given the values of the other variables that **con** applies to.

5. If any variables have changed in Step 4, update the dependency sets of these variables to include the dependencies of the other variables and **con**.

6. For all constraints that act on variables changed by the above operation, if these constraints are not in **constraints** add them back to that list.

7. If **constraints** is empty, exit, else go to Step 2.

Figure 3.5: Constraint Propagation Algorithm

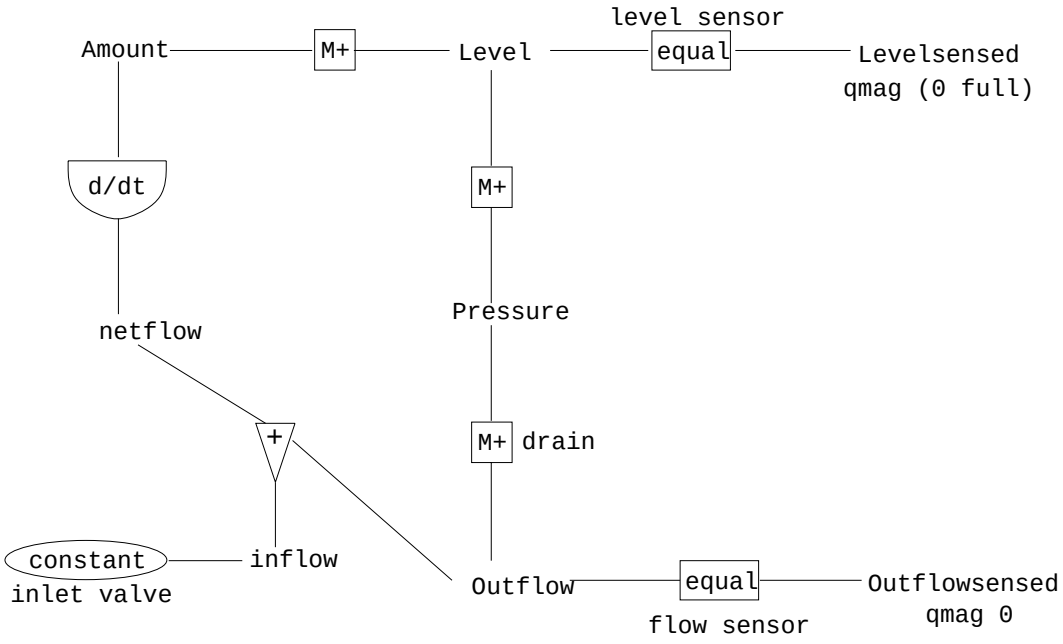

Figure 3.6: Bathtub Constraint Network

also causes the dependencies on the variable `Level` to get set to the singleton set containing the component mode value `(levelsensor-mode normal)`.

In this way, propagation continues from `Levelsensed` and from `Outflowsensed` until the contradiction between the two values is detected at some constraint. The exact point at which the contradiction is detected is dependent on the ordering of the constraints. As an example, consider that the propagation process has determined that `pressure` must be in the interval `(0 high)` (where `high` is the pressure corresponding to a `full` value for `Level`) and its dependency set is {`(levelsensor-mode normal)`}. The `Outflow`, getting its value from `Outflowsensed` is `0` with a dependency set of {`(flowsensor-mode normal)`}. Now the constraint `(M+ Pressure Outflow)`, which is associated with the `drain` is checked and found to be inconsistent because `Outflow`'s value, `0`, must correspond, according to the constraint's corresponding values, to a `0` value to a `pressure` of `0`, instead of the `(0 high)` that we computed. This causes the dependencies of this constraint to be updated and the resulting dependency set to be returned as a conflict. The dependency set in this case is the union of the dependency sets of the variables involved in the contradiction along with the component mode value associated with the constraint. Thus the conflict set would be {`(levelsensor-mode normal)`, `(drain-mode normal)`, `(flowsensor-mode normal)`}.

3.3.2 Constraint Satisfaction

If the propagation procedure described above concludes with a conflict set, QDOCS exits from the hypothesis checker and returns the computed conflict. On the other hand if the propagation procedure completes the state by assigning a value to every variable in the system, again, the hypothesis checker concludes that the state is consistent and exits. However, as is often the case, if the propagator computes a partial set of values for some of the variables of the system and is blocked in its attempts to propagate values any further, as we discussed in Chapter 2, QDOCS needs to use a more general constraint satisfaction algorithm to complete states. As with propagation, we borrow the outline of the main algorithm from QSIM, but we incorporate dependency maintenance into the algorithm.

The constraint satisfaction algorithm can further be divided into two parts: the first is a Waltz-filtering algorithm [Waltz, 1975], while the second is an exhaustive backtracking over the space of possibilities. Figure 3.7 is the top level description of QSIM's version of the Waltz filtering algorithm, with our modifications. The basic loop consists of going through the list of constraints and filtering out the tuples that violate the constraints. Whenever one or more tuples is eliminated from a constraint, the propagation algorithm of Figure 3.8 is used to transmit the effects of the filtering out to the other variables and constraints of the constraint network. As with the propagation algorithm, dependency maintenance occurs during the course of the propagation of effects of constraints and follows the definitions at the beginning of this section.

If the Waltz-filtering completes successfully and does not find a contradiction, there is still a possibility that there is a contradiction. This possibility arises because Waltz-filtering merely establishes that there is at least one possible value for each variable that is compatible with the other neighbouring constraints in the constraint network. However, there may not be any single, consistent, global assignment of values to all the variables.

Thus we still need to revert to a backtracking procedure to complete the constraint satisfaction process. The algorithm QDOCS inherits from QSIM for this follows the standard backtracking methodology - values are first assigned to each variable that has only one possible value, and then one of the set of possible tuples is assigned to each constraint and the corresponding variable values are assigned to the relevant variables. Whenever a variable has no possible value, that branch of assignments fails and the procedure backtracks. A complete failure occurs when all branches have been exhausted and no consistent assignment of values to variables is found.

In order to return a valid conflict set in case of this kind of failure, QDOCS needs to maintain temporary dependency sets for each branch of the backtracking. These dependency sets are compiled by taking the union of all the dependency sets for each variable along the branch of the search computed during Waltz-filtering. Every time a branch fails, it records that failure as being due to all the components in the dependency sets of all the

QSIM-**Waltz-filter**

1. Initially set list **constraints** to all the constraints in the model.
2. Set the **pvals** of each variable to be all the possible values given the result of propagation.
3. Set the **tuples** of each constraint to be the cross product of all the possible values of the variables it acts upon.
4. Choose the *most restrictive constraint*, or the one with the smallest number of possible tuples, from **constraints** and remove it from the list.
5. Apply the constraint to remove all tuples that are inconsistent with the constraint. Update its dependencies set to include the component mode values associated with the constraint as well as the dependencies of all the variables the constraint acts upon.
6. Starting from this constraint, propagate its effects out by calling **Propagate-Out** (constraint). If the value returned is a conflict, exit, returning this conflict.
7. If some constraint has no possible tuples, or if some variable has no possible values, return its dependency set as a conflict, otherwise go to Step 4.

Figure 3.7: The top level of the Waltz Filtering algorithm

Propagate-Out (Constraint)
For each variable **var** that **Constraint** acts on, if the number of possible values for this variable has been reduced,

1. Update the dependencies of this variable to include all the dependencies of **Constraint**. If the variable has no possible values, exit, returning these dependencies as a conflict from the top level call to **Propagate-Out**.

2. For all other constraints acting on **var** if the number of tuples associated with the constraint must be reduced because of **var**, update their dependencies to include those of **var** and call **Propagate-Out** on all the other constraints acting upon **var**.

Figure 3.8: Propagating effects out in the Waltz-filtering algorithm

constraints and variables assigned thus far. The conflict set would then be the union of all the dependency sets of all the failed branches of the backtracking. Failures in this part of the constraint satisfaction process are rare but when they do occur, the fact that we need to do an exhaustive accounting of all the dependencies of all constraints involved in every failed branch means that the conflict sets generated will usually be large.

To illustrate the constraint satisfaction algorithm, consider the set of constraints shown in Figure 3.9. The quantity spaces and constraints shown are intended to be a part of a larger overall system. The part of the constraint network shown has five variables, each with quantity spaces consisting of three landmarks – 0, var_0 and var_1, where var must be substituted by the name of the variable. There are three relevant constraints on these variables, with the first two being addition constraints stating that $A + B = C$ and $C + D = E$. The last constraint shown says that B increases monotonically with E. The constraints listed also show the relevant corresponding values for the constraints. For the first constraint $0 + b_0 = c_0$, $a_0 + 0 = c_0$, and so on. Similar corresponding values hold for all the other constraints. Each of the constraints given is also assumed to be associated with a different component - `Comp1`, `Comp2`, and `Comp3` respectively.[1]

In the particular example we will consider, A is known to have the value a_0 and D is known to be d_0.[2] The top diagram in Figure 3.10 shows the network corresponding to these

[1] As we have seen QDOCS actually maintains the mode values of these components in its dependency sets, but in this example, for simplicity, we show just the components in the dependency sets.

[2] QDOCS will also consider qualitative directions in doing constraint satisfaction, but we omit these in this example for simplicity.

Quantity Spaces:

A: {0 a_0 a_1}
B: {0 b_0 b_1}
C: {0 c_0 c_1}
D: {0 d_0 d_1}
E: {0 e_0 e_1}
...

Constraints:

((+ A B C) (0 0 0) (0 b_0 c_0) (a_0 0 c_0) (a_0 b_0 c_1))
((+ C E D) (0 0 0) (0 e_0 d_0) (c_0 0 d_0) (c_0 e_0 d_1))
((M+ B E) (0 e_0) (b_0 e_1))
...

Figure 3.9: Constraints for Waltz-filtering example

constraints. Since propagation cannot transmit much information across a + constraint given a value for only one of the variables, none of the possible values for the variables B, C, and E, are ruled out. Each of the constraints initially has possible tuples corresponding to the entire cross-product of the possible values of each of the variables. This amounts to 25 tuples for each of the three constraints given.

The Waltz-filtering algorithm now applies each of these constraints to keep only those that satisfy the constraints, and then propagates the effects out to all the other variables and constraints in the system. Since each of the constraints currently has the same number of possible tuples, any of them may be selected for this step. Suppose the constraint $A + B = C$ is selected next. The second diagram in Figure 3.10 demonstrates the effects of this. Only three tuples survive the application of this constraint and the Waltz-filter will propagate this effect to other variables and constraints in the network. The only values of variable B that survive are the values less than b_0 while the only values of variable C that survive are the values greater than c_0. These in turn affect the possible tuples of the other constraints acting on them and thus the other two constraints are down to 15 tuples each. The component `Comp1` gets added to the dependency sets of each of these variables and constraints as shown in the Figure.

In the next step, the Waltz-filtering algorithm may apply the other + constraint. Since C is now down to values equal to or higher than c_0 and since D is known to be d_0, the only tuple that survives has C being c_0 and E being 0. The effects of this are again propagated. The first + constraint now has just one remaining tuple, B has 0 as its only value and the M+ constraint has the tuple [0,0] as its only possible tuple. All of these have `Comp2` added to their dependency sets. The final step of the algorithm (not shown)

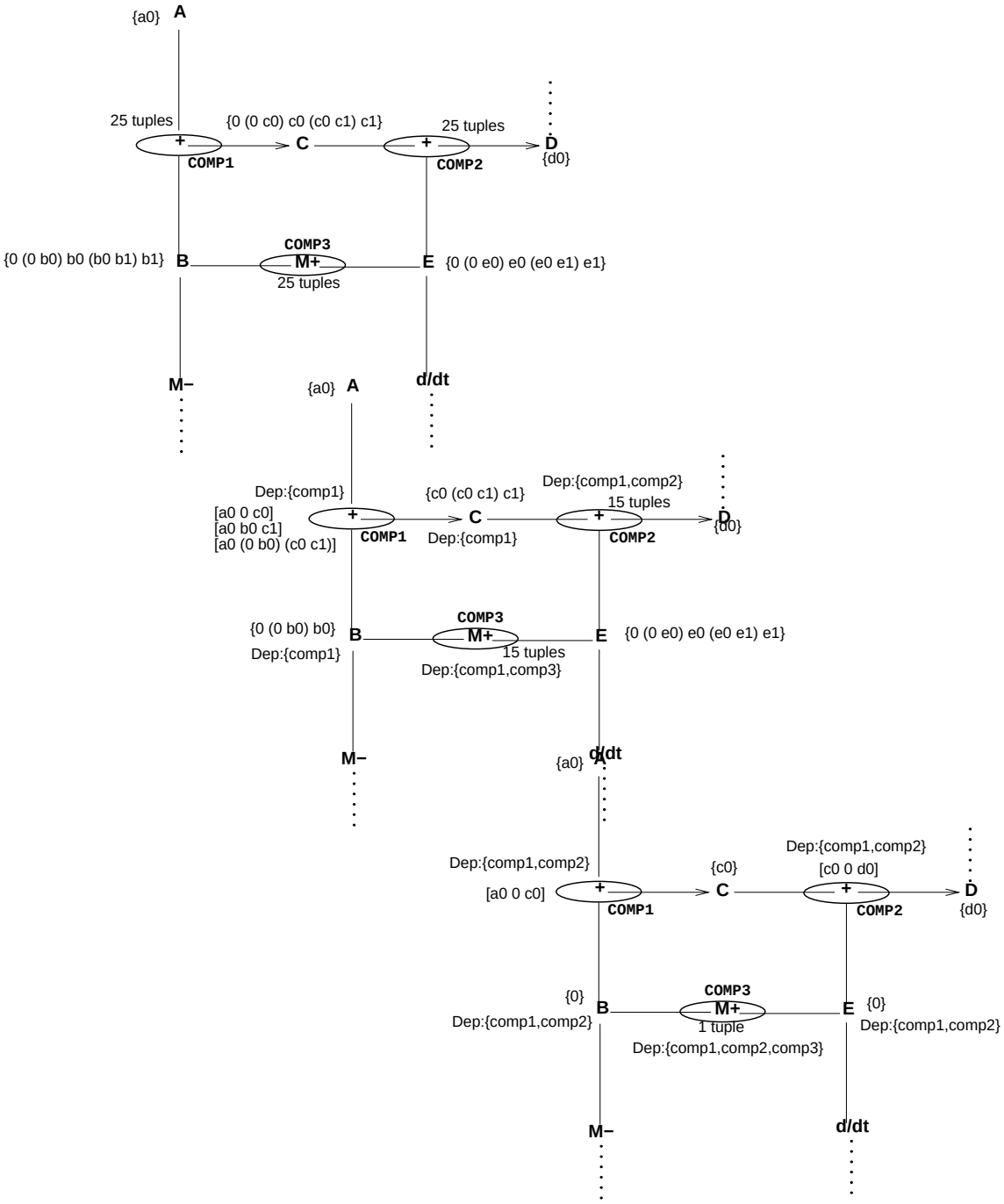

Figure 3.10: Waltz-filtering example

attempts to satisfy the M+ constraint. Since the only remaining tuple [0,0] does not satisfy the constraint, there is a contradiction. The conflict returned is {`Comp1, Comp2, Comp3`} since the dependency set for this constraint includes all of these components.

Note that this procedure does not guarantee minimal conflict sets. If for example, the algorithm had applied the M– constraint on B and that had reduced the number of possible values for B, B may well have had some other dependencies that would have been a part of the conflict set even though the three constraints we used are contradictory.

Minimality of conflict sets can only be achieved through some sort of exhaustive analysis of the constraint network. When an inconsistency is detected, the analysis would have to determine exactly why each possible value for the variable was ruled out. This would be a far more computationally explosive process than our current algorithm.

The effect of non-minimality of conflict sets is simply to slow down the diagnostic process. Since each conflict may have component mode values that are not necessarily part of a minimal conflict set, the **Diagnose** algorithm of Figure 3.4 will generate and test many more inconsistent hypotheses than a procedure that guaranteed minimal conflicts would. However, the soundness of the diagnostic procedure is not affected since any superset of a conflict set is still a conflict set and any correct diagnosis will still hit that larger conflict set.

3.3.3 Across-Time Verification

The first phase of the hypothesis checking process is verification that each of the individual states of the given behavior is consistent with the hypothesis. If this phase succeeds in its verification task (and thus fails in the conflict generation task), the second phase involves verification that there is a simulation of the model corresponding to the hypothesis which matches the observed behavior.

The key concept involved in determining if there is a simulation corresponding to a given behavior is continuity. The sequence of states must be continuous and at each step the states must have qualitative values for sensor values that match the sensor values given in the behavior. QDOCS tackles this problem in the following manner. Starting with the first state, it uses the completions generated by the constraint satisfaction procedures above and proceeds to generate successors using QSIM, keeping only the ones that match the sensor values in the given behavior. This continues until either the last state is reached or there are no possible states matching a particular sensor reading.

There is one problem, however, that makes this matching much harder than it might seem. For any given set of sensor readings, there can be an arbitrarily long chain of possible states that will match the given set while being different in qualitative values of quantities that are not among the sensor values. This fact causes the search space of possible states to combinatorially explode and makes the search intractable.

It is important to note, however, that this is due to a basic problem with the intractability of some QSIM simulations. Since the models of some systems are inherently underconstrained under certain fault modes, QSIM can sometimes produce a large number of very similar successors, differing only in that certain underconstrained variables take on various different values. When QSIM has a large branching factor on a particular model, QDOCS can also have a large number of possible successor states that it must try and match.

In order to keep computation times under control, QDOCS uses two techniques to speed up search. First, the search is carried out in a depth-first manner, i.e., whenever a successor of a previous state matches the next set of sensor readings in the behavior, it is chosen to be expanded before the ones matching the current set of sensor readings. The second simplification used is that the number of states allowed to match a single set of sensor readings is limited.[3] This has the effect of making QDOCS unsound in principle since the correct hypothesis may not be matched, but our experiments never revealed a case where this was a problem – as we shall see in Chapter 5, in all our experiments, a model containing the correct set of faults (or a subset of these faults) was always able to match the given behavior.

The problem of determining conflict sets if the simulation fails to match the readings is a tricky one. In Chapter 4 we report on our techniques for doing this. However, as we report there, the extensions used to determine conflict sets in these problems tended to include all or most of the component mode values, and it was thus usually much more efficient to simply return a default overly-general conflict set consisting of all the component mode values of the hypothesis being tested. As we discussed in the previous section, an overly-general (or non-minimal) conflict set simply has the effect of slowing down the diagnostic process without affecting the soundness of the diagnoses.

3.4 Analysis of QDOCS algorithm

We have shown in Theorem 3.2.1, that the top level search algorithm of QDOCS always gives the most probable consistent hypothesis given a sequence of observations. However this result was dependent on the correctness of the **Check-Hypothesis** procedure. We have also shown (Theorem 3.3.1) that any procedure that maintains the definitions of dependency (Definitions 3.3.2 and 3.3.3) is guaranteed to produce correct conflict sets. Since our constraint propagation and constraint satisfaction algorithms update dependencies according to these definitions, Theorem 3.3.1 guarantees that if **Check-Hypothesis** returns with a conflict generated by these procedures, the conflict set is a correct one.

The last step in **Check-Hypothesis** is the across-time verification algorithm. If the verifier successfully matches the given behavior, it is guaranteed that the behavior

[3]The limit is 3 in all the experiments reported on in Chapter 5.

corresponds to a QSIM simulation of the given hypothesis. If there is no QSIM simulation corresponding to the given behavior, the across-time verifier will fail and return an overly-general or non-minimal conflict set as we mentioned previously.

The only reason we cannot guarantee the soundness of QDOCS, however, is because our limit of 3 consecutive states per set of sensor readings means that the across-time verifier can fail to match behaviors that may be valid QSIM simulations of the given model, and thus produce incorrect conflict sets. As we mentioned before, however, this limitation was only inserted for efficiency reasons.

Without this limitation, if we allow the across-time verifier to generate consecutive sequences of states of arbitrary length, it is true that any QSIM generated behavior of the given model will be verified by QDOCS as a correct simulation of the model. This is because the across-time verifier works by attempting to match the given states in the observed behavior with QSIM states generated by considering all valid successors of the initial state. Since by definition, the observed sequence must correspond to a branch in the behavior tree generated by simulating the model corresponding to the best diagnosis, QDOCS will succeed in matching the behavior.

As we mentioned before, in all our experiments, a hypothesis with a subset of the "correct" set of faults was always able to match the given sequence of observations. When this happens, QDOCS does not even attempt to match the superset or correct set of faults. If this correct set were not matched because of the 3-state limitation, QDOCS would generate hypotheses with lower probability than the correct set of faults. Since, as we shall see, this never happened in our experiments, we may conclude that the limitation on consecutive matches has no practical effect on the soundness of QDOCS in our particular domains. In Chapter 7, we suggest alternative approaches to the across-time verification problem that may avoid this limitation while preserving soundness.

Thus we have shown that given a sequence of observed sensor values corresponding to a behavior of the modeled system, QDOCS will find the most probable hypothesis that has a QSIM behavior corresponding to the given observations. What we have not shown is that this method is practical to implement and use, or that all of its parts are needed to be able to diagnose faults in any realistic applications. In Chapter 5, we will empirically test QDOCS on these criteria. We have implemented a prototype of the QDOCS system as described here in Common Lisp. It runs on top of an implementation of the QSIM system [Kuipers, 1994]. All of our experiments were performed on a Sun Sparcstation 5 running Lucid Common Lisp.

Chapter 4

Diagnosis Algorithm Extensions

In Chapter 3 we outlined the basic procedures that constitute the QDOCS system. In this chapter, we will outline some of the extensions we have implemented to either speed up the QDOCS implementation, or to extend its capabilities. As we report later, these efforts have achieved mixed success, and we will discuss the failures as well as the successes of these efforts.

4.1 Handling Region Transitions

A device will often have a number of different modes in which it can operate even under normal conditions. For example, if we added an overflow drain to the bathtub example in Chapter 3, as shown in Figure 4.1, we would have two different regions of normal behavior for the device. If the water level was below the level of the overflow drain, there would only be one drain for the bathtub. On the other hand, if the water level were above the point of the overflow drain, there would be two points from which the water would drain.

In simulating such a device, QSIM handles having multiple models by establishing conditions under which *region transitions* take place and by positing a new set of constraints that go into effect after the transition. In QDOCS we use the same functions, but we also require the modeler to declare the possible regions that a component can be in under different possible modes. For instance, with a bathtub that has an overflow drain, we might have a component called `overflow-drain` that would have two different regions. The following would be the associated constraints:

```
(mode (and (overflow-drain-mode normal)
           (overflow-drain-region overflowing))
      ((M+ level overflow-rate)))

(mode (and (overflow-drain-mode normal)
```

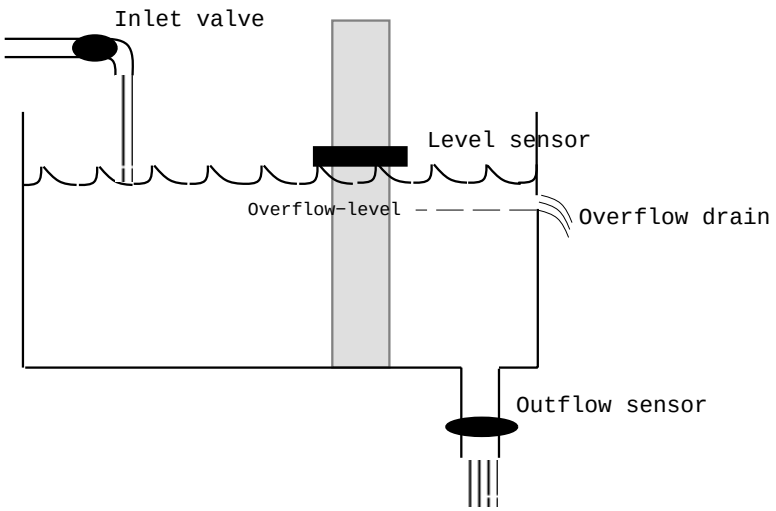

Figure 4.1: A Bathtub with an Overflow Drain

```
      (overflow-drain-region inactive))
 ((ZERO-STD overflow-rate)))
```

In these constraints, the first mode variable in the condition, `overflow-drain-mode`, is the component mode variable for the overflow drain component. Under its normal operation, it can be in one of two regions – `overflowing` or `inactive`. When it is `overflowing`, the drain rate through the overflow channel monotonically increases with the level, while when it is in the inactive region, there is no flow through this channel.

QDOCS needs a couple of extensions to be able to diagnose faults in devices with multiple operating regions. First, it needs an exhaustive declaration of which operating regions are available within which modes. This declaration looks like the following:

```
(defOperatingRegions extended-bathtub-model
   ((overflow-drain-mode . normal)
    (overflow-drain-region . overflowing)
    (overflow-drain-region . inactive)))
```

The declaration tells QDOCS that when the `overflow-drain-mode` is `normal`, the `overflow-drain-region` must either be `overflowing` or `inactive`, but not both. QDOCS will thus be able to test hypotheses in which this mode variable is `normal` under each of these assumptions to see if the observations are compatible with either mode.

There is, however, one other extension required for the QDOCS model to be able to diagnose models with operating regions. Under particular operating regions, certain assumptions regarding the values of other variables are implicit in the QSIM model. For

example, in the overflowing bathtub model, it is not possible for the overflow drain to be in overflowing mode when the level of water is lower than the overflow level. This is unimportant for simulation in QSIM because the conditions on region transitions ensure that there will never be a situation where the level is lower than the overflow level with the region variable in overflowing mode. However, when we are testing hypotheses in QDOCS these conditions need to be made explicit so that we can test individual states without information about which region the state is supposed to be in.

QDOCS allows this by extending QSIM's constraint language to include *integrity constraints*. These are constraints such as the following:

```
(mode (and (overflow-drain-mode normal)
           (overflow-drain-region overflowing))
      ((greater-than-or-equal level overflow-level)))
(mode (and (overflow-drain-mode normal)
           (overflow-drain-region inactive))
      ((less-than level overflow-level)))
```

These constraints require that the `level` be higher than the landmark `overflow-level` when the `overflow-drain-region` is in the `overflowing` region and be lower than the landmark otherwise. Integrity constraints work like any other QSIM constraints and have been integrated into the version of the QSIM system we use for our experiments.

The approach taken by the hypothesis checker when the model has multiple operating regions is simply to generate all the possible combinations of operating regions and try each of them in turn for each of the sets of sensor readings. If any of the combinations of operating regions has consistent completions, the hypothesis is considered to be consistent with that set. If, on the other hand, they all fail, the union of all the conflict sets returned is returned as a conflict set.

In Chapter 5 we will look at two different domains in which QDOCS has been tested which have multiple operating regions. The results on these domains demonstrate the workability of these techniques.

4.2 Extending Across-Time Verification

As we mentioned in Section 3.3.3, the present version of the across-time verification algorithm as implemented in QDOCS simply attempts to simulate the given behavior using QSIM and if it fails, returns the entire hypothesis as a conflict set. This would seem to not quite be an ideal approach and conflicts with the general approach to the diagnosis problem we espoused in Chapter 2.

The general approach taken in QDOCS is to attempt to explain the failure of QSIM to produce the given behavior as a simulation by analyzing which constraints are responsible

for blocking such a simulation. Following this approach, our algorithm for across-time verification was to track variable dependencies as we were simulating across time and to have successors inherit these dependencies from predecessor states. When some branch of the behavior tree fails to have any consistent successor, it is tagged as a failing branch and the conflict set returned by the constraint satisfaction algorithm for that branch is computed.

The final conflict set returned was the union of all the conflict sets associated with the failing branches at the deepest level of sensor readings that could be matched. The argument behind returning this as a conflict set was that the reason it is impossible to simulate the entire behaviour is because it is impossible to extend any simulation that goes through to the previous set of sensor values, to the following set of sensor values.

This approach was abandoned for a number of reasons:

- First, the space and time requirements for tracking all these dependencies were prohibitive for some of the domains studied.

- Second, because the typical branching of QSIM behaviors is quite high, the conflicts produced typically contained a lot of components anyway, thus making the conflict sets less useful.

- Third, the argument above supporting the computation of conflict sets is not entirely correct. Suppose that a certain model is able to simulate a behaviour up till the third set of sensor readings but no farther. The above algorithm would only look at components that caused the model to fail to reach the fourth set of readings from the third set. However, the constraints due to some component may have prevented the simulation from establishing certain values for certain variables in a state matching, say, the third set of sensor readings, that would then have made it possible to simulate to the fourth set and beyond. In other words, every failing branch is a source from which we must derive a conflict set whose combined union is the conflict set to be returned. This process would get prohibitively expensive.

These issues are discussed again in Chapter 7 where we propose alternative approaches that may result in efficient ways to derive conflict sets from the failure to simulate a model across time.

4.3 Caching

One of the bigger sources of inefficiency in the QDOCS algorithms is the repeated application of the same constraints to the same variable values as a number of slightly different hypotheses are checked. This prompted us to consider using some form of *dependency caching* in

order to try to reuse computation from the testing of previous hypotheses while checking later ones.

We were inspired in devising a dependency caching scheme by the research on Truth Maintenance Systems(TMS) [Doyle, 1979; de Kleer, 1986]. The TMS system that is most relevant to our problem is the Clause Management System (CMS) of Reiter and de Kleer [1987]. The CMS in turn was a generalization of the more popular Assumption-based TMS (or ATMS) of de Kleer [1986]. As we mentioned in Chapter 2, the ATMS maintains *justifications* which are supporting sets of beliefs that support a belief. Some of these supporting beliefs are *assumptions* that by definition justify themselves. The ATMS compiles these into *labels* for each belief where a label is a set of minimal *environments*. An environment in the label of a belief is simply a set of assumptions whose conjunction supports that belief.

The ATMS, however, makes the assumption that justifications are always Horn-clauses. In other words, they always have a conjunction of assumptions implying a single proposition. The CMS, on the other hand, is a generalization of this scheme that is designed to maintain dependencies for *clauses* or disjunctions of propositions. The ATMS is useful in systems like GDE [de Kleer & Williams, 1987] which use constraint propagation, since each propagation step has a conjunction of values implying a single consequence. However, since QDOCS's hypothesis checker maintains disjunctions of variable values associated with sets of assumptions, the CMS scheme would be the one to use. QDOCS's dependencies look like the following:

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \longrightarrow (V = v_1) \vee (V = v_2) \vee \ldots (V = v_m)$$

where each A_i is either a mode variable value or a sensor value from the state on which the hypothesis is being tested, and each v_j is a possible value for variable V.

The CMS is required to always compute minimal labels for any clause for which it is maintaining supports (i.e., ones for which any subset of assumptions in the label would not be sufficient to imply the clause). The minimality is modulo the information it has at any given time. This means that new justifications may have the effect of turning a minimal label into a non-minimal one.

The most general way to maintain such labels is by using resolution among the support clauses to determine the minimal label. QDOCS, however, has certain regularities that make the computation of such minimal labels much more tractable than it would be with a general resolution theorem prover. In order to make this clear, we shall start with the following lemma that follows from the monotonicity of constraint application.

Lemma 4.3.1 *If* $A = \{A_1, A_2, \ldots, A_n\}$ *and* $B = \{B_1, B_2, \ldots, B_m\}$ *are sets of assumptions (mode values as well as sensor values), and* $P_1 = \{(V = x_1), (V = x_2), \ldots, (V = x_k)\}$ *and* $P_2 = \{(V = y_1), (V = y_2), \ldots, (V = y_l)\}$ *are sets of possible values for variable* V*, then*

if $\bigwedge A \longrightarrow \bigvee P_1$ [1] *and* P_2 *is the smallest set such that* $\bigwedge B \longrightarrow \bigvee P_2$ *and* $A \subset B$*, then* $P_2 \subseteq P_1$.

In other words, for any variable, a larger set of assumptions necessarily means a fewer set of possible values. The assumptions here are simply the dependency sets as defined in Section 3.3, extended to include sensor values.

Since QDOCS computations often do not produce the smallest set of possible values for a given dependency set, the dependency caching scheme will attempt to use an efficient representation to determine if, given a set of possible values and a dependency set derived from the constraint satisfaction algorithm, whether the dependency set is minimal at the time for that set of possible values. If it is minimal, it will add information to the cache, and would thus be worthy of caching. In general, if the computed set of values P for some variable has a dependency set S, it adds information if every existing cache entry with a dependency set that is a subset of S (possibly improper), has a set of possible values that is not a subset of P. In other words, the cache does not already have information that would allow QDOCS to restrict the set of possible values to a set smaller than P with a smaller set of assumptions.

Typical accesses into the proposed cache are upon initialization of the constraint network, prior to constraint propagation and satisfaction. Given a hypothesis, we look for the most restrictive set of values that we can determine for the variable, given the assumptions in the hypothesis and the sensor values. These values and their dependencies are installed in the constraint network and the hypothesis checking algorithms proceed as before.

The mechanism we provided for caching with accesses as given above is a directed acyclic graph (DAG) for each variable. Each node in the DAG, n has an associated dependency set $n.dep$ and an associated set of possible values, $n.pv$. The root node of the DAG has the empty dependency set and its set of possible values has all the landmarks and ranges in the quantity space for the variable with all possible qualitative directions.

Anywhere, in the DAG, if there is a link from node n_1 to node n_2 (i.e., if n_2 is a *child* of n_1), the following conditions must be satisfied:

$$n_1.dep \subset n_2.dep$$

$$n_1.pv \supset n_2.pv$$

Thus as one goes down the dependency graph, the nodes have dependency sets that are supersets of their parent's dependency sets, and sets of possible values that are subsets of their parent's pv sets. Figure 4.2 is a simple example of what such a cache might look like.

[1] $\bigwedge X$ refers to the conjunction of all the elements of set X while $\bigvee Y$ refers to the disjunction of all the elements of set Y.

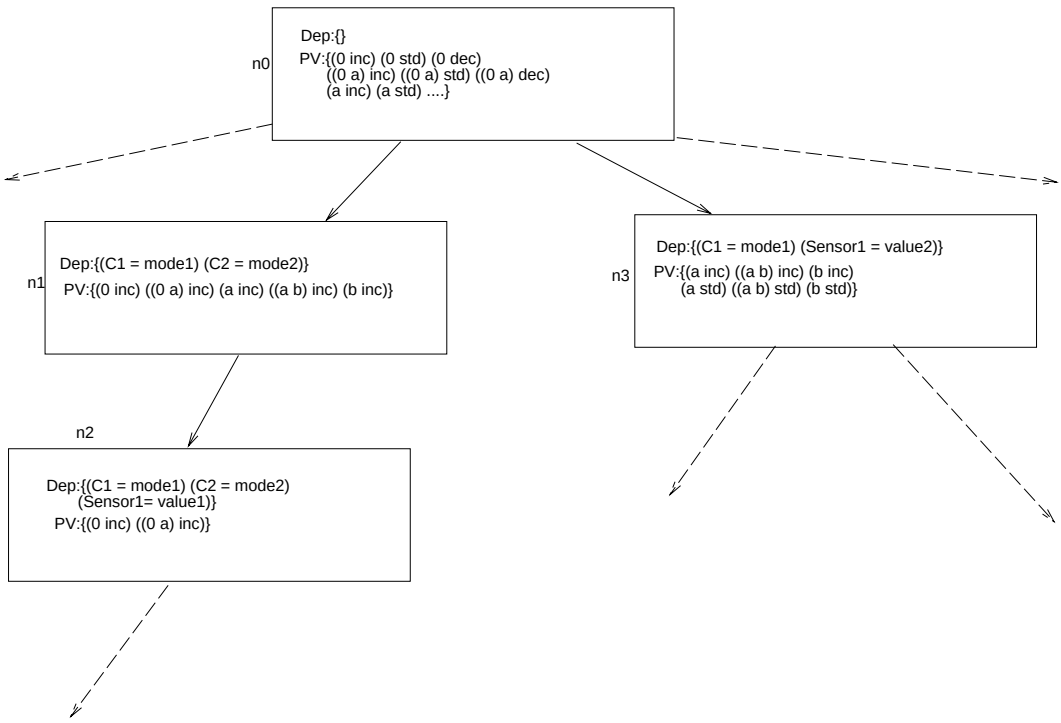

Figure 4.2: An Example Cache

The access algorithm examines the cache for each variable and install a set of values and a corresponding dependency set. Given the hypothesis (augmented with sensor values) the algorithm goes down each branch with dependency sets that are subsets of the hypothesis and retrieves the deepest nodes with such dependency sets. The installed values are the intersection of the pv sets for these nodes and the dependencies are their union.

As an example, suppose we have the cache shown in Figure 4.2 for a particular variable and we are given a hypothesis consisting of the component values $(C1 = mode1)$ and $(C2 = mode2)$ and the particular state we are testing has the sensor $(Sensor1 = value2)$.

First, node $n0$ is considered. The root node has an empty dependency set and thus is always matched. Next each of the children is considered and each child with a dependency set that is a subset of the current set is considered for traversal. Since $n1$ is matched, it is traversed next. Next the procedure looks at each of $n1$'s children and discovers that the only child, $n2$ has a dependency set that is not a subset of the hypothesis. Thus $n1$ is the deepest node on this path.

Next, the node on the other branch, $n3$, is examined. It, too, has a dependency set that is a subset of our hypothesis. Assuming then that none of its children satisfies this condition, the nodes $n1$ and $n3$ are returned as the deepest nodes whose dependency sets are subsets of the hypothesis.

Now since we know that

$$n1.dep \longrightarrow n1.pv$$

and

$$n3.dep \longrightarrow n3.pv$$

then we also know that

$$n1.dep \cup n3.dep \longrightarrow n1.pv \cap n3.pv$$

since the dependency sets are a conjunction of literals and the pv sets are a disjunction of possible values. Now since both $n1.dep$ and $n3.dep$ are subsets of our hypothesis, $n1.dep \cup n3.dep$ must also be a subset of the hypothesis. Thus we can restrict our set of possible values to $n1.pv \cap n3.pv$ and give it the dependency set of $n1.dep \cup n3.dep$.

The algorithm for maintaining the given properties of the cache while adding new elements is a little more complex but still fairly efficient. Given a dependency set and a set of possible values, the caching procedure must find all points on the DAG where the new entry can be inserted and check that the new entry does, in fact, add information to the graph. If it does, a node is added or changed, and the effects are propagated downwards to ensure that the conditions on the edges in the graph are maintained.

This approach to caching dependencies met with limited success in our experiments as we shall see in Chapter 5. A discussion of this scheme follows the presentation of the results.

Chapter 5

Experimental Evaluation

In previous chapters we have described the algorithm, and its implementation in QDOCS, for the diagnosis of dynamic systems modeled qualitatively using QSIM. We have made some claims about the approach that we will attempt to verify empirically. Among these are:

1. QDOCS, because of its ability to detect inconsistencies and generate conflicts in situations where constraint propagation is blocked, is more accurate than a pure propagation approach like INC-DIAGNOSE,

2. Because of QDOCS's ability to focus in on diagnoses through the use of conflict sets, it is a faster approach than the baseline generate-and-test method mentioned in Section 2.3, and

3. Each of the distinct phases of QDOCS's hypothesis checking algorithm contributes either to improve the accuracy or the efficiency of the QDOCS algorithm.

The following section introduces the methodology we use to test QDOCS and verify these claims. In the two sections following this, we introduce two domains in which we have tested the QDOCS system. We present the problems studied, a description of our models of these systems, and then follow these by the results of our experiments. Finally, we report on the results of an experiment to test the utility of the caching scheme we introduced in Chapter 4.

5.1 Experimental Methodology

In this section, we will look at the methodology we used in our experiments on our two test domains – the Reaction Control System of the space shuttle, and a level-controller for a reaction tank. Since we were unable to obtain real data to test our methods, we relied on

data that was generated by simulating the actual models. Faults were randomly generated using the probabilities given in the component model to determine their distribution. The result is a number of different sets of faults, some where all components are `normal`, some single faults and some multiple faults.

In order to further limit our search to those problems that would be hardest to diagnose, i.e., multiple fault scenarios, we further picked out only those sets of faults which had at least two components faulty.[1] The models corresponding to these faults were simulated using QSIM from a consistent initial state with other variables all set to reasonable initial ranges. The system is then simulated until some predefined state limit is reached.

The result of the QSIM simulation is a behavior tree out of which the experiment generator chooses a path at random. A path in the tree is a sequence of states corresponding to a qualitative behavior of this faulty system. The experiment generator then extracts the sensor readings from each of these states and merges consecutive states which have identical sensor values. The resulting sequence is then the sequence of sensor readings as seen by the monitoring system.

These resulting behaviors may still not be very interesting from a diagnostic standpoint. Very often, a faulty system will still produce a behavior that is indistinguishable from a normal one. For example, if there is a leak somewhere in the system, it is still possible for the leak to be small enough that it makes no qualitative difference in the behavior of the system. Also, some parts of these systems we study are built just for fault-tolerance. Therefore, to narrow our experimentation to the interesting problems, we chose to restrict our experiments to those behaviors which can only be produced by a QSIM simulation of a model of the given system containing at least two faults.

Some of these eliminated behaviors may correspond to system behaviors that are actually faulty but where the multiple faults could not be detected using qualitative reasoning alone. A more complete accuracy test of our system would include a simulation of a numerical model to check if there is actually a numerical deviation from the normal model (or a single-fault model). Such a test would allow us to gauge the overall accuracy of our methods. However, because numerical models were difficult to build for these systems, we restricted ourselves to the qualitative model and only used behaviors that showed qualitatively distinguishable failures.

We contend that the behaviors that we eliminate in this process are simply uninteresting from the standpoint of multiple-fault diagnosis. Since they are behaviors that could have been produced by a no-fault model or a single-fault model, any qualitative diagnostic system should come up with a no-fault or single-fault hypothesis to explain them as these are usually more probable than a multiple-fault scenario.

[1] We also ran experiments on single-fault problems. While QDOCS performed well in those problems and was better than the generate-and-test approach, the advantage of QDOCS was not as pronounced as with the more computationally expensive multiple-fault problems.

QDOCS is compared to various different techniques and for each of these we collect data on the efficiency and accuracy of the methods. First, a generate-and-test method is used as a baseline comparison. This technique uses the same hypothesis checker as QDOCS without all the bookkeeping required to generate conflict sets, but simply tests hypotheses generated in most-probable-first order. Note that given the fact that QSIM (and hence QDOCS) makes acausal inferences, a generate-and-test procedure is the best we can do without using QDOCS-style dependency propagation.

We then also compared QDOCS with a number of ablated versions in order to justify all the different parts of the hypothesis checker. First, it was compared against a system that simply used QDOCS's constraint propagation procedure. This version is equivalent to INC-DIAGNOSE, enhanced to be able to diagnose with behavioral modes.

Another ablated version of QDOCS we test against is one with both the propagation and constraint satisfaction parts of the code but without the across-time verification code. This test is to determine if the across-time verifier (which is one of the most computationally expensive parts of QDOCS) is worthwhile in improving the accuracy of the system.

Finally, we test a version of QDOCS that used the constraint satisfaction and across-time verification portions of the hypothesis checker but skipped the constraint propagation portion. This comparison was run to verify that that the constraint propagation algorithm speeds up the constraint satisfaction process even though the constraint satisfaction and across-time verification algorithms together are just as powerful (in terms of accuracy of diagnoses) as the complete QDOCS hypothesis checker.

An assumption we made regarding the overall framework in which QDOCS is to be used, is that the user or monitoring system that calls our diagnostic program is interested in a range of the most probable hypotheses consistent with the observations. It would then use this information to implement recovery methods such as making further measurements or replacing certain components. Therefore on each problem, QDOCS was run until the best remaining hypothesis had a probability of less than a tenth of the probability of the best (i.e., most probable) hypothesis that was found to be consistent with the observations thus far. This would give us a range of all the consistent hypotheses that were within an order of magnitude of each other in *a priori* probability and would provide a termination condition for the top-level procedure of QDOCS. In each of our domains, we first generated a test suite of 100 examples and ran the above experiments on all of them.

5.2 Reaction Control System

The first problem we look at is that of diagnosing faults in the Reaction Control System (RCS) of the Space Shuttle. The RCS is a collection of jets that provides motion control for the orbiter when it is in space. These jets are fired appropriately whenever changes need

to be made to the orientation or position of the craft. The main tasks of the RCS include separation from the external fuel tank after launch and reorienting the shuttle for reentry into the earth's atmosphere. We will first describe the RCS system itself and how we have modeled it and then summarize the results we obtained.

5.2.1 The RCS Model

The entire RCS system consists of three separate subsystems, two with 14 jets each and one with 16. The two with 14 jets are located in the rear of the craft to the left and the right, while the subsystem with 16 jets is located in the nose of the craft. Their positions are shown in Figure 5.1. Each subsystem contains a fuel tank and an oxidizer tank. The fuel and oxidizer are mixed and combustion takes place in the thruster which then feeds the jets.

Figure 5.2 shows the complete layout of an RCS subsystem. As shown in the diagram, there are two nearly identical but independent parts of the subsystem, meeting only at the thrusters. These parts deliver, respectively, the fuel and the oxidizer, to the thruster chamber.

Each of these parts works by using helium to pressurize the propellant.[2] The helium feeds through a system of pressure regulators, into the propellant tank where it maintains a steady pressure so that the propellant exits the tank at a steady rate. The propellant then gets channeled into one of many manifolds from which it exits into the corresponding thruster.

Figure 5.2 shows two different paths between the helium and the propellant tank. These are provided to make the system reconfigurable. At any point, only one of these paths is kept open. Each of the paths has two pressure regulators, one being redundant and used for fault-tolerance. Also, the lines between the propellant tank and the manifolds are also linked to crossfeeds that allow sharing of propellant with the other subsystems of the RCS.

A QSIM model for this system was first built by Kay [1992]. This model has been extended and modified by us for the purposes of diagnosis. Still, the modeled system is a somewhat simplified version of the actual RCS subsystem, as shown in Figure 5.3. One simplification we make is to consider only a single path between the helium and propellant tank, since only one is active at any one time. We also abstract away the multiple manifolds into a single component with one value for pressure. Since the manifolds are interconnected and maintain one pressure anyway, this is a reasonable abstraction. Finally, we assume the presence of eight sensors to monitor the system - pressure sensors in the Helium tank, the propellant tank and the manifold, as well as propellant volume sensors in the propellant tanks.

[2] The word *propellant* here refers generically to the fuel or the oxidizer.

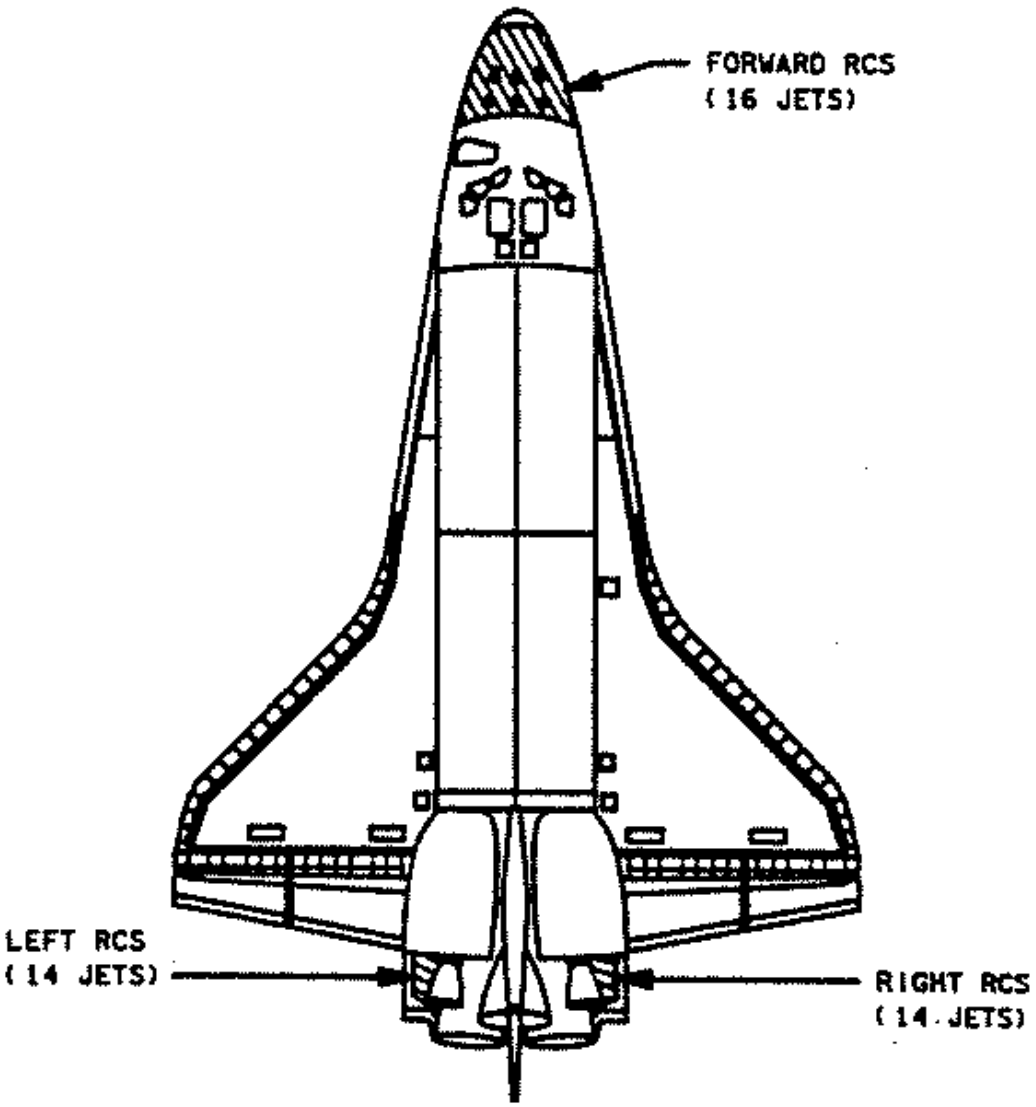

Figure 5.1: Position of RCS subsystems

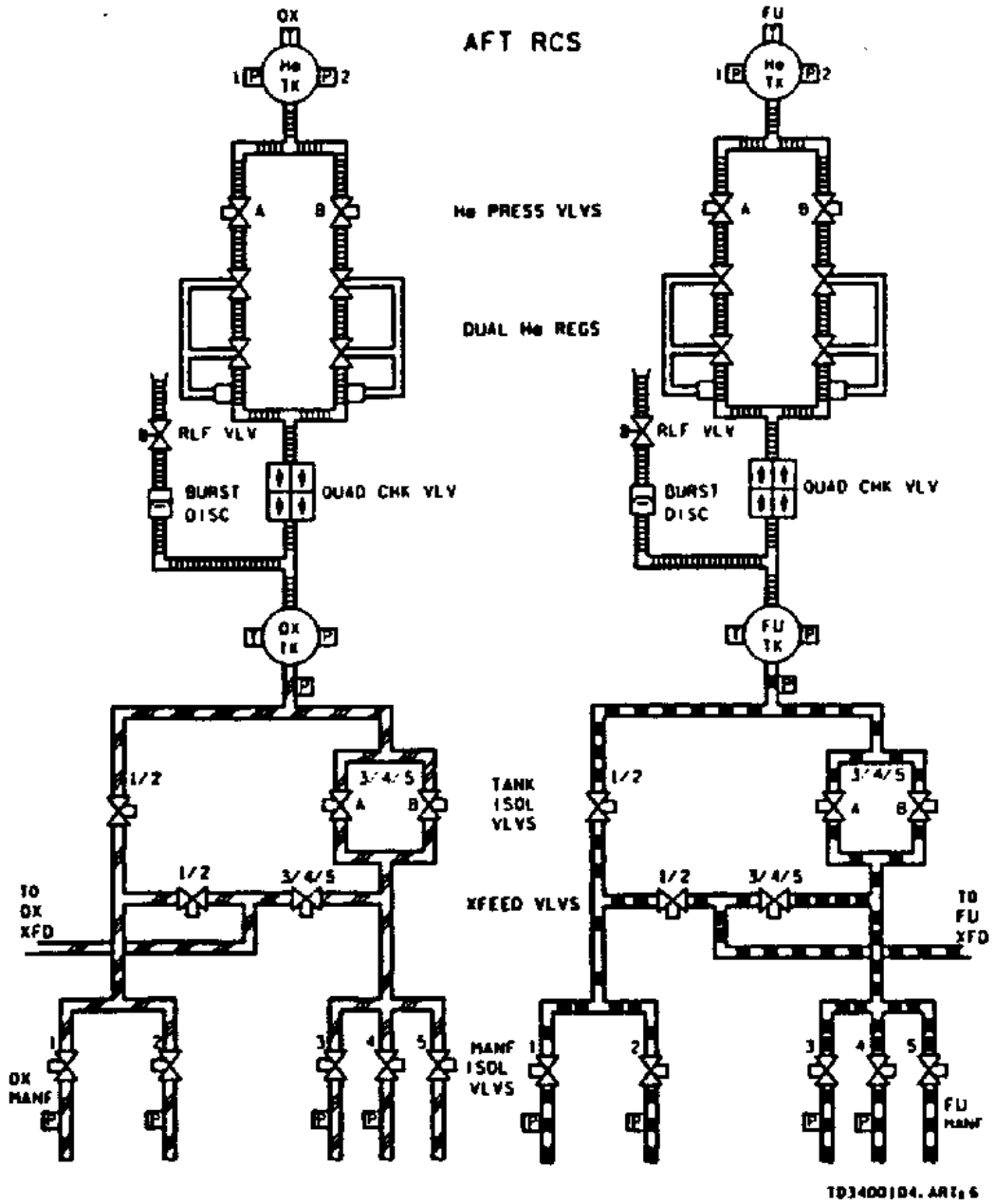

Figure 5.2: Layout of an RCS Subsystem

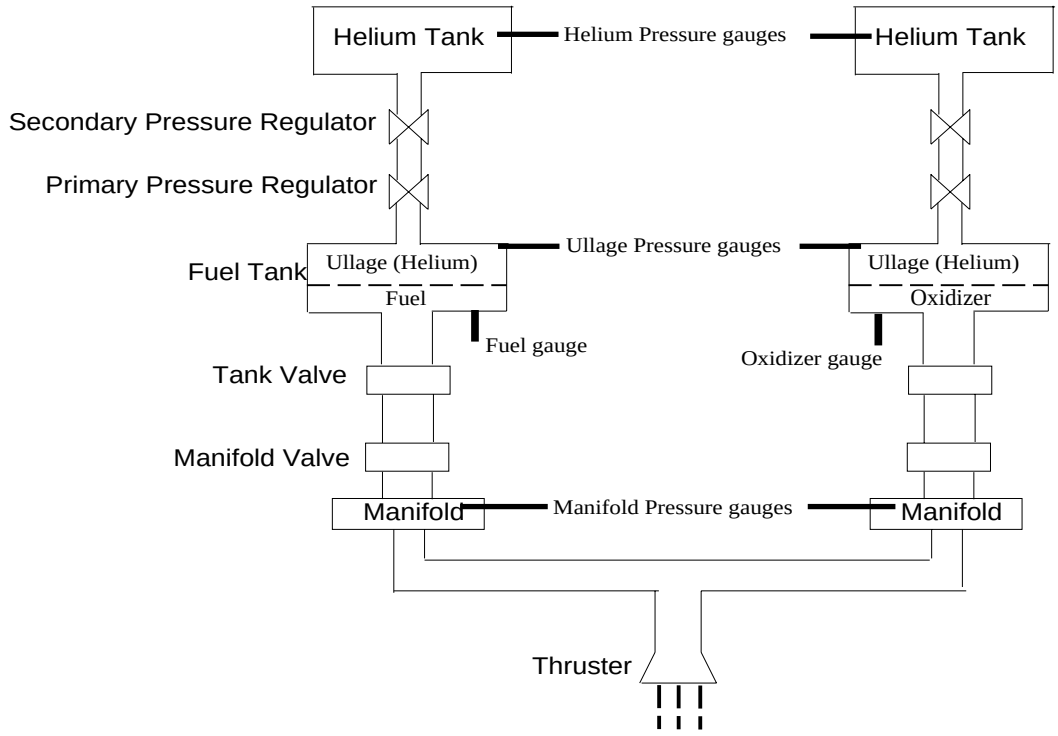

Figure 5.3: A Simplified RCS Subsystem

The QSIM model and associated QDOCS component structure for this system are given in Appendix A. The complete model contains 135 constraints and 23 components, each with multiple behavioral modes. Some of the kinds of faults modeled include pressure regulators stuck open and closed, leaks in the helium tank, the fuel tank, or the fuel line, and sensors being stuck low or high.

The pressure regulators are modeled as ideal regulators. In other words, if the input side of the regulator is at a higher pressure than a certain setpoint, we assume that a normally functioning regulator will maintain the setpoint on the output side. The regulator also operates in multiple regions within the normal mode – when the input pressure is above the setpoint, it behaves ideally, while when it is below the setpoint, it acts like a pipe, so that the same pressure is maintained on both sides of the regulator.

Thus the RCS model also requires a listing of operating regions within modes and these are also listed in Appendix A. Since we have four pressure regulators with 2 regions each, there are a total of 16 different region combinations that are possible if all the regulators are functioning normally. If they are functioning abnormally, they are either stuck open or stuck closed and hence their behavior is completely determined.

The difference between the pressure of Helium in the propellant tank[3] and the pressure of propellant in the manifold determines the rate of flow of the propellant. The propellant enters the manifold, which is simply treated as a container of pressurized fluid, and then from there flows out into the thruster. The higher the pressure in the manifold, the faster the flow out into the thruster.

The different leaks affect the system differently in terms of where the effects are felt. Leaks in the helium tank are felt in the helium system as well as in the propellant tank, where the propellant is at a lower pressure, and thus flows out very slowly. This would in turn, imply a lower pressure in the manifold. Leaks in the fuel line, on the other hand, are only felt in the manifold and in a faster drain rate from the propellant tank. Lastly, manifold leaks affect only the manifold pressure (and are affected by the manifold pressure) and again increase the drain rate from the propellant tank.

The two pressure regulators can break in open or closed positions. Since they are placed in sequence, it is often impossible to distinguish between faults in them. For example if the primary regulator is working normally and the secondary regulator is stuck closed, the the model is identical to that obtained when the primary regulator is stuck closed but the secondary regulator is normal.

Other possible faults mainly involve problems with the sensors used to gauge the system. Each of the sensors could either be stuck low or high. These faults obviously only affect the readings from the particular sensors, and they are detected when the other sensors show normal behavior while the faulty sensor is constant and either low or high.

[3]The Helium in the propellant tank is also known as the *ullage*.

Method	Avg. # Hyps.	Most Prob. Correct %	Most Prob. Subset %	Member Correct %	Member Subset %	Run Time (sec)	Hyps. Tested
Gen & Test	1.39	77.00	95.00	82.00	100.00	1288.77	456.55
Prop. only	2.91	29.00	82.00	32.00	85.00	44.39	19.71
No Across	1.71	42.00	84.00	42.00	84.00	85.68	25.29
No Prop.	1.39	77.00	95.00	82.00	100.00	647.95	52.62
QDOCS	1.39	77.00	95.00	82.00	100.00	454.02	52.70

Figure 5.4: Results in the RCS domain

Since the actual probabilities of the faults were unknown, they were assigned by us with normal modes being much more common than the fault modes. As with all QDOCS models, we make the assumption that the faults are independent of each other. A sample run of QDOCS on a problem in this domain is shown in Appendix B.

5.2.2 RCS Experiments

We ran the series of experiments described in Section 5.1 on the RCS system. The results are summarized in Figure 5.4. As mentioned in Section 5.1, each method was run on the 100 generated test cases and all consistent hypotheses were maintained until the next possible hypothesis (or the most probable remaining hypothesis) is less than 0.1 times as probable as the hypothesis computed thus far with the highest probability. The first column reports the average number of hypotheses generated per diagnosis problem for each of the tests. The second through fifth columns show different measures of accuracy for each method, while the last two columns show different measures of efficiency.

For each method we separated out the most probable hypotheses (often more than one if there were a few equally probable hypotheses) and compared these to the correct hypothesis. The percentage of cases where the correct hypothesis was among these is reported in the second column. In many cases, a subset of the correct faults is sufficient to model the given behavior. The percentage of cases where a hypothesis which is one of the most probable is a subset of the correct set of faults is shown in the third column. The fourth column shows the percentage of cases for which the right hypothesis occurs anywhere in the set of hypotheses generated while the fifth shows the percentage of cases in which some hypothesis is a subset of the faults of the correct hypothesis.

The sixth column shows the average time taken for each problem on a Sparc 5 workstation running Lucid Common Lisp. Finally, the last column has the number of hypotheses the hypothesis checker actually had to test.

The first line shows the results obtained using the generate and test method. When we compare this to the complete QDOCS algorithm on the last line, we see that they both

have identical accuracies – 82% of problems are accurately diagnosed in both cases, with 77% of the correct solutions also being among the most probable. In all the cases a hypothesis had at least a subset of the faults of the correct hypothesis. This result is as expected since a systematic elimination of hypotheses as in the generate and test method is guaranteed to reach the right hypotheses eventually. The big difference appears in the average number of hypotheses tested – the generate and test method tests 8.7 times more hypotheses than QDOCS.

What this shows is that QDOCS is able to narrow the hypotheses considerably using its dependency propagation algorithms. However, the run times show that there is a cost that QDOCS pays for this advantage. The ratio of run times is 2.8 to 1 in favor of QDOCS. The difference between the ratios for hypotheses tested and for time is explained by the difference in the amount of bookkeeping that QDOCS must perform to keep track of all the dependencies. This is still a substantial advantage for QDOCS over the simpler generate and test method and clearly demonstrates its superiority. As we discuss in Chapter 7, there is scope for improvement in efficiency in the area of dependency updating, and thus the ratio of run times could improve further.

Figure 5.4 also shows that the different parts of the QDOCS hypothesis checker contribute either to the accuracy or the efficiency of QDOCS. The second line of the table shows the results obtained by running the same problems using just the constraint propagator as the hypothesis checker. As expected, the accuracy results are low compared to QDOCS since there are many conflicts which simply cannot be detected by the propagator alone. However, since propagation is a very efficient procedure, the average times are only about 44 seconds instead of the 454 seconds that the full QDOCS uses.

When constraint satisfaction was added to the propagation procedure (line 3 or "No Across"), the accuracy improved, but the method is still not quite as accurate as the complete QDOCS algorithm (42 % vs. 82 % for QDOCS). This shows that the across-time consistency checking is useful in pruning incorrect hypotheses. Again, however, the technique took much less time on average than the QDOCS algorithm. The difference here (86 secs. vs. 454 secs.) is explained first by the difference in the average number of hypotheses tested (25.29 vs. 52.70) as well as by the fact that the across-time verification process is computationally expensive.

Another measure of the accuracy of these methods comes from comparing the hypotheses they generate with the best hypotheses that can be generated to explain the behavior. Since QDOCS is guaranteed to return the most probable hypothesis consistent with the observations (from the results of Chapter 3), it scores 100% on this measure. The comparison on this measure between QDOCS and the two ablation tests from Figure 5.4 that have lower accuracies than QDOCS is shown in Figure 5.5. Note that "Propagation only" now has an accuracy rate of just 38% (i.e., in 38% of the cases, the most probable consistent

Method	"Best" Hypothesis %
Prop. only	38.00
No Across	55.00
QDOCS	100.00

Figure 5.5: Comparison to best hypothesis

Method	Avg. # Hyps.	Most Prob. Correct %	Most Prob. Subset %	Member Correct %	Member Subset %	Run Time (sec)	Hyps. Tested
Small RCS/ Gen &Test	1.76	86.00	98.00	88.00	100.00	263.87	145.60
Small RCS/ QDOCS	1.76	86.00	98.00	88.00	100.00	220.76	33.12
Full RCS/ Gen & Test	1.39	78.00	93.00	82.00	100.00	1288.77	456.55
Full RCS/ QDOCS	1.39	78.00	93.00	82.00	100.00	454.02	52.70

Figure 5.6: Results on RCS subproblem compared with full RCS

hypothesis was also among the most probable hypotheses selected by the method) while QDOCS scores a full 100% on this measure. On this same measure, "No Across" scores 55%, thus showing that both constraint satisfaction and across-time verification improve the accuracy of the QDOCS procedure significantly.

Finally, when we ran QDOCS without the constraint propagator (line 4 of Figure 5.4), we found that the system was just as accurate as the whole QDOCS. This is because the Waltz-filtering algorithm of QSIM can make all the inferences that the propagator can. However, this method was also considerably slower, showing that the propagator was indeed useful in increasing the efficiency of QDOCS.

Another interesting experiment we conducted regarding run time comparisons between QDOCS and the generate and test method was a study of a part of the RCS subsystem consisting of a single propellant flow path to the thruster. The model for this system is almost exactly half the size of the full RCS subsystem model. We generated problems in the same way as for the experiments reported on in Figure 5.4, and ran 100 problems through the generate and test and QDOCS algorithms. Figure 5.6 shows a comparison between the results on this subproblem with the result of the RCS problem considered above. Again, the accuracies were identical (88% correct, 100% subset) between the two methods. However, the run times averaged 264 seconds for the generate and test and 221 for QDOCS. This

is a ratio of only 1.2 to 1 even though the ratio of hypotheses tested was 4.4 to 1. The corresponding ratios for the complete system are 2.9 to 1 and 8.7 to 1. This suggests that for similar problems, the larger the problem size, the greater the advantage of using a dependency propagation algorithm like QDOCS to generate conflict sets. We therefore expect the advantages of QDOCS to be greater for even larger problem sizes.

These experiments show that for multiple fault diagnosis on the RCS, QDOCS is clearly superior to the methods listed. In the following section, we shall apply the methods to a very different problem to determine if the QDOCS method is applicable and gets similar results on an unrelated domain.

5.3 Level-Controlled Tank

One very common system in chemical engineering applications is a reaction tank with an outflow and some system to control the level to keep it constant. Various versions of controlled tanks have been studied for simulation and monitoring using QSIM (e.g., [Catino, 1993]). Most such studies have concentrated on how to simulate particular controlling functions for the level controller. Our study, on the other hand, will concentrate on diagnosing faults in such a feedback system. To do this, we model the level-controller in greater detail and assume monitoring sensors in the electrical circuitry.

Figure 5.7 shows our representation of an implementation of a level controller for a tank. The controller works as follows. As the level of the water in the tank ($h(t)$) rises, the float also rises, causing the contact point on the resistor to rise. This higher setting on the resistor will set up a higher potential difference across the amplifier which in turn multiplies the difference with a fixed *gain* value. The resulting potential is then applied to a motor which in turn controls the valve. Water flowing through the valve flows directly into the tank. For higher values of the error ($h(t) - setting$), the angular velocity (ω_m)of the valve becomes higher, except that ω_m becomes 0 when the angle of opening of the valves (θ_m) reaches either 0 or `thetammax`.

This system is taken from a standard control systems textbook, [Kuo, 1991], and is known to be an unstable system that can display wildly oscillating behaviors. However, the instability plays little role in the ability to diagnose faults in this domain. The main reason this system is of interest is to show that the QDOCS mode of dependency propagation is useful even for feedback systems. Some researchers (e.g., [Dvorak & Kuipers, 1992]) have held that such an algorithm would not be useful in dynamic systems with feedback loops because variable values are usually dependent on all constraints. However, if we have the ability to monitor particular values in the feedback loop, we break the loop into parts in such a way that everything is not dependent on everything else.

The level-controlled tank is modeled as shown in Appendix C using a QSIM model

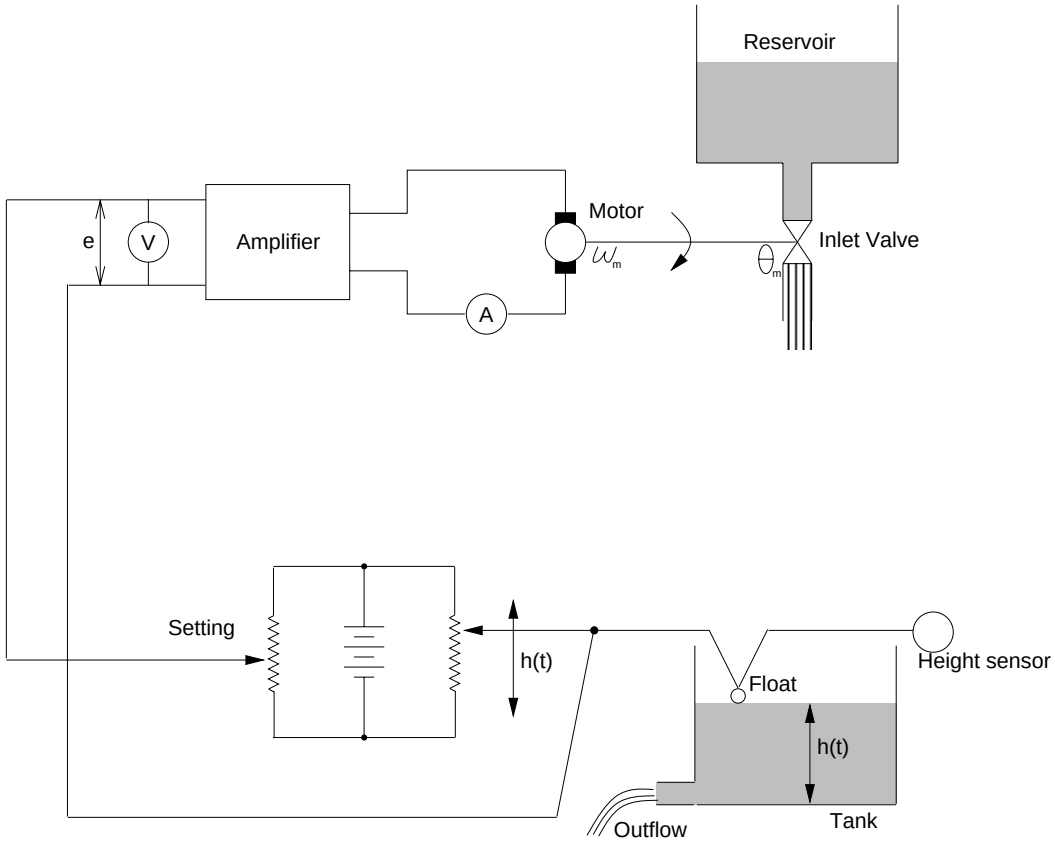

Figure 5.7: A level controller for a tank

Method	Avg. # Hyps.	Most Prob. Correct %	Most Prob. Subset %	Member Correct %	Member Subset %	Run Time (sec)	Hyps. Tested
Gen & Test	4.96	50.00	58.00	88.00	98.00	59.21	215.14
Prop. only	4.63	39.00	68.00	65.00	99.00	6.63	26.60
No Across	4.65	39.00	67.00	65.00	99.00	7.69	26.83
No Prop.	4.96	50.00	58.00	88.00	98.00	31.50	24.31
QDOCS	4.96	50.00	58.00	88.00	98.00	27.39	33.95

Figure 5.8: Results in the Level-Controller domain

Method	"Best" Hypothesis %
Prop. only	78.00
No Across	78.00
QDOCS	100.00

Figure 5.9: Comparison to best hypothesis

with 45 constraints and a component structure with 14 components. The various possible faults include leaks in the tank, a clogged drain, breaks in the electrical circuitry, a dead battery, a stuck voltmeter or ammeter, a stuck motor, and valves that are either stuck open or closed.

We ran all the experiments described in Section 5.1 on this model of the controlled tank. Again the focus was on multiple faults, and we restricted ourselves to those problems that required a multiple-fault diagnosis. As before, the experiments were produced by randomly generating and simulating 100 faults in the QSIM model. The results are summarized in Figure 5.8.

As in the equivalent experiments with the RCS, QDOCS does better than the other techniques. It is about 2.2 times faster and tests 6.3 times fewer hypotheses than the generate and test method. QDOCS is also more accurate than either propagation alone or propagation and constraint satisfaction (no across-time). The results of comparing these methods in terms of the most likely consistent hypotheses is shown in Figure 5.9. This, as for the RCS, shows that both "Prop Only" and "No Across" are significantly less likely than QDOCS to find the most probable hypothesis that is consistent with the observations.

Another interesting result in Figure 5.8 is that the less accurate methods actually had marginally higher percentages in the Member Subset column. This demonstrates the limitations of subset percentages. Given that the correctness accuracies of these methods were lower, the only conclusion we can make about them is that certain subset hypotheses

were matched by these methods which failed on across-time verification in QDOCS. This shows that the subset results may actually reward methods that match incorrect hypotheses, and is thus not an entirely reliable measure of accuracy.

Another major difference with the RCS results was hat the QDOCS version without the propagator had an accuracy that was almost equal to QDOCS (as expected), but was only a little slower. We believe this is because the smaller constraint network of the level-controlled tank means that propagation does not provide as much of a speedup over the exhaustive constraint satisfaction system as it did on the RCS problem. A related result was that propagation and constraint satisfaction performed almost identically as propagation alone. This is probably again because propagation is not much more efficient than constraint satisfaction on this smaller set of constraints.

5.4 The Utility of Caching

In this section, we study the utility of the preliminary work on caching of dependencies that we introduced in Chapter 4. One of the main problems with any caching scheme is the possibility that the overhead associated with the caching scheme may be higher than the speedups that it makes possible. This is often referred to as the *utility problem* (a detailed study of the problem has been undertaken by Minton [1988]) and it was to study this phenomenon that we performed an experiment to break up the costs associated with the hypothesis checking and the cache updating.

Unfortunately, the caching scheme, as currently implemented in QDOCS, does seem to exhibit characteristics of the utility problem, although it does show enough promise to be considered worthy of continued study. In the best cases, caching did marginally speed up the operation of QDOCS, while in the worst ones, the slowdown was fairly significant.

The caching scheme is designed to speed up the operation of the constraint propagator and the constraint satisfaction algorithms, during the phase when QDOCS is checking consistency within a single qualitative state (as opposed to during the across-time verification stage). Because the same constraint satisfaction functions are called from both the single-state completion phase and the across-time verifier, the experiments we used to test the caching scheme were all conducted with the across-time verifier turned off. This was necessary to truly gauge the time the program was spending doing constraint satisfaction in the phase in which caching is supposed to reduce the amount of computation to be peformed.

We make the assumption that diagnostic episodes occur frequently and thus it would make sense to keep a cache over a number of runs of the diagnosis system. However, one problem with caching across a number of episodes is that caches can grow exceedingly large as each call to the constraint satisfaction algorithm can yield a slightly different set of

Method	Prop. Time (per call) sec	CSP Time (per call) sec	Update Time (per call) sec	Access Time (per call) sec	Calls per prob.	Avg. Time per prob. sec
No Caching Prop+CSP	0.056	0.23	NA	0.045	45.70	21.22
Caching Prop+CSP	0.020	0.17	0.043	0.160	61.00	31.17
No Caching CSP only	NA	0.34	NA	0.046	63.60	32.65
Caching CSP only	NA	0.20	0.026	0.140	63.35	30.76

Figure 5.10: Results of Caching experiments

possible values for each variable. This set, as long as it meets the criteria for caching given in Chapter 4, will add to the cache size even if the specific case is not particularly useful. In Chapter 7, we will suggest some ways to minimize cache size, but for these experiments, we simply use a numeric limit on the number of nodes in the cache (set to 30 in all these cases) and stop caching inferences for a given variable when this limit is reached in the cache for that variable.

The experiments were run with the smaller RCS model (i.e., half of the RCS subsystem) over 20 problems each. The results are summarized in Figure 5.10. All the problems were run on the same set of randomly generated double faults. We used Lucid Common Lisp's MONITOR facility to record the times spent in each of the routines of interest as well as to count the number of calls to these routines. The first and second lines of Figure 5.10 show the results of running the propagation and constraint satisfaction algorithms, while the third and fourth lines show the results of running only the constraint satisfaction algorithm.

The first four columns display the average time that QDOCS spends in each call of each of the particular functions shown. The first column shows the amount of time spent on average per call to the propagator, while the second shows the time spent per call to the constraint satisfaction algorithm. The third and fourth columns show the average times spent in each call of the cache updating and cache accessing functions respectively.[4] While using both the propagation and constraint satisfaction algorithms (rows 1 and 2), the propagation time per call fell by more than half (from 0.056 secs. to 0.020 secs.) while the constraint satisfaction (CSP) time fell by a significant amount (from 0.23 secs. to 0.17 secs). However the cache access time is somewhat higher for the caching scheme than the

[4]In the case of the non-caching algorithms, access times are actually the times it takes to compute the space of possible values for each variable given an initial partially specified value. This operation is performed in the caching schemes as part of the cache access.

non-caching scheme by a factor of 3.5.

An unexpected result was that there was a large difference in the number of calls to the constraint propagation and satisfaction functions between the different problem solving methods. The number of calls was significantly higher with caching than without. We believe that caching has the effect of inflating the size of dependency sets.

The reason for this is as follows: suppose some variable v is where a contradiction is detected for a particular hypothesis. In cases where we are not using a cache, the variable would be initialized with no dependencies and all possible values. During the process of constraint propagation and constraint satisfaction, it gets component mode values added on to its dependency set, while the number of possible values decreases to zero. The dependency set that remains at that time is a conflict set.

On the other hand, when we are using a cache, the variable may start off with a dependency set from a previously cached computation with elements that may not be necessary to cause a conflict at that variable. The constraint satisfaction algorithm would, however, include those component mode values as part of the conflict. While the process of constraint satisfaction may be speeded up because the set of initial possible values would be smaller due to the cache, the result is a conflict set with more component mode values. As we have pointed out before, larger conflict set sizes usually mean slower diagnosis. This problem, along with the high costs of updating and access seem to be the major factors in explaining the poor performance of caching.

Significantly, when we ran the constraint satisfaction algorithm alone without the propagator, there was a small speedup with the caching algorithm, when compared to the algorithm without caching. In this case, the number of calls to the CSP actually drops (but not significantly), while the cache access and cache update times are low enough that the faster CSP times still give caching an advantage. The problem with inflated caches is not as pronounced here. This is because most of the gain in terms of small conflict sizes tends to be in the constraint propagation algorithm and while that advantage was lost when moving to constraint satisfaction alone, the disadvantage of initializing variables with dependencies from the cache becomes less significant since dependency set sizes tend to be large in any case.

Thus in conclusion, caching seems to speed up the actual time taken per call to the constraint satisfaction algorithm, but the costs of that speedup outweigh the benefits in the system as currently implemented. In Chapter 7 we will look at ways in which we can overcome these problems in the future.

Chapter 6

Related Work

In the preceding chapters, we have outlined the QDOCS approach to addressing the problem of qualitative model-based diagnosis of continuous dynamic systems. As we saw in Chapter 2, our particular approach was motivated by some of the shortcomings of some of the precursors to QDOCS. Some of the main distinctive features of the system we built include:

1. It diagnoses with a qualitative description of a dynamic system, whereas most previous efforts at model-based diagnosis have been with static systems.

2. It incorporates a general multiple-fault diagnostic system using behavioral modes.

3. It computes conflict sets for use in diagnosis using a constraint satisfaction algorithm that works for cases where the more commonly used constraint propagation algorithms fail.

In this chapter, we will study the relationship between the QDOCS approach to diagnosis and a number of other approaches that have been studied in the literature. We begin by reviewing the approaches that have already been introduced in Chapter 2 and then move on to some of the other related systems. We collected related work into groups of systems that use roughly the same approach.

6.1 INC-DIAGNOSE and DIAMON

Ng's INC-DIAGNOSE system [1991] has already been discussed in some detail in Chapter 2. It's main limitation is the fact that it is restricted to a subset of QSIM constraint networks where variable values can always be propagated across constraints unless there is a contradiction. Since this is only true of a small subset of possible constraint networks, this is a severe limitation. In contrast, as discussed earlier, QDOCS is able to work with any

model that can be simulated using QSIM. Thus it is more generally applicable than INC-DIAGNOSE. Furthermore, as we pointed out in Chapter 2, a constraint propagation system that is incomplete is likely to produce diagnoses that are incorrect, since inconsistencies in hypotheses tested may simply not be detected.

Another limitation of INC-DIAGNOSE is that, as implemented by Ng, it is unable to take advantage of known failure modes (or behavioral modes) of a device. However, as has been shown by various researchers [Struss & Dressler, 1989; de Kleer & Williams, 1989], extending Reiter's basic formulation of the diagnosis problem to include fault modes is not difficult.

INC-DIAGNOSE was only tested on one very small system (a proportionally-controlled thermostat) and for only a small set of behaviors of that system. The limitations mentioned would have prevented its application to most realistic systems.

Another system that uses an INC-DIAGNOSE style diagnosis system is DIAMON [Lackinger & Nejdl, 1991]. The diagnosis subsystem in DIAMON,[1] like INC-DIAGNOSE, performs constraint propagation using QSIM constraints, but it also adds a hierarchical focussing system on top of the propagation layer. The hierarchical focussing system allows the propagator to *zoom in* on specific parts of the constraint network that may be faulty by analyzing coarser descriptions first and identifying broad areas that may be faulty. This use of hierarchies in diagnosis is actually an older idea dating back to [Genesereth, 1984]. However, like INC-DIAGNOSE, DIAMON is limited to domains in which constraint propagation is sufficient to determine all the conflict sets.

6.2 MIMIC and QMIMIC

As we saw in Chapter 2, MIMIC [Dvorak, 1992] is primarily a monitoring system that uses a simple dependency tracing algorithm on a causal influence graph to determine the possible fault hypotheses when an error is detected. We pointed out two main problems with the MIMIC approach. They are:

1. Since reasoning in QSIM can occur in an acausal manner, the policy of looking only at faults that are upstream from the sensor where the fault is detected is unsound, and

2. MIMIC makes a single-fault assumption, and is thus unable to diagnose multiple faults.

One feature that MIMIC has, and that QDOCS does not, is its ability to make use of semi-quantitative information. Qualitative landmark values for variables are described using ranges of possible numeric values that allow for more precise discrepancy detection. However, these semi-quantitative values do not play a role in the actual fault generation

[1] DIAMON integrates such a diagnosis system into a monitoring framework.

process because MIMIC essentially discards this information and looks solely at the causal dependency graph to generate hypotheses. As we discuss in Chapter 7, for a QDOCS-like diagnosis system to be able to work with a semi-quantitative monitoring system like MIMIC, we will need to adapt it to use this information to generate hypotheses.

MIMIC was applied to a number of fairly simple examples. A very similar system that was used on some more complex problems is QMIMIC [Vinson & Ungar, 1995]. This system is an enhancement to MIMIC that allows for better handling of sensor noise. This capability along with other enhancements in the monitoring system allow it to be used in complex applications like the monitoring and diagnosis of a propylene glycol reactor. However, it uses essentially the same diagnosis algorithm as MIMIC and thus suffers from the limitations enumerated above.

6.3 MAGELLAN-MT, DOC, and CA-EN

In this section we consider three of the more recent systems that address the problem of diagnosis of continuous physical systems. All of these, as we shall see, use different techniques to test hypotheses and construct conflicts. However, they must make restrictive assumptions about the nature of their diagnosis problems in order to be able to model them in such a manner as to make such computation possible and efficient.

The first of these systems is MAGELLAN-MT [Dressler, 1994]. This system tackles the diagnosis of continuous dynamic systems by using an efficient default-logic based formulation which was introduced in [Dressler & Struss, 1992] and extended in [Dressler & Struss, 1994]. The basic diagnosis engine of MAGELLAN-MT uses a focussing mechanism based on applying default logic to a partial ordering of hypotheses to generate new candidate hypotheses. Hypotheses are then tested in a focussed manner only on portions of the device where the faults are likely to predict values which are different from previously computed default values.

On the problem of consistency across time, Dressler claims that most conflicts can be detected using single time slices alone without doing any across time conflict generation. This claim agrees with our experience in QDOCS and the systems on which we tested. Like QDOCS, MAGELLAN-MT uses the across-time simulator mainly to verify hypotheses rather than generate conflicts. Since, as we have seen with QDOCS, across-time simulation is clearly more expensive than single-state conflict detection, this shows that our approach of checking hypotheses on individual states first and verifying across time only if necessary, is the most efficient way of testing hypotheses and generating conflicts.

At the basic level, however, MAGELLAN-MT uses a simple ATMS to keep track of dependencies. Since the ATMS is unable to reason with disjunctions of propositions, this means that there is an implicit assumption that constraints or logical rules in the system

will lead to specific values for variables. Thus this system suffers from the same limitations as INC-DIAGNOSE and other constraint propagation systems we have discussed. However, the alternative diagnostic formalism and the focussing heuristics employed are worthy of further study and may be applicable to a more general constraint satisfaction system such as the one used in QDOCS. The authors claim high accuracies and speeds in diagnosing faults in real-time in a ballast tank system for a large ship.

Another recent system that tackles the problem of diagnosis in continuous dynamic systems, is DOC [Kapadia, Biswas, & Robertson, 1994]. DOC makes two assumptions regarding the problem it is solving. First, it assumes that the system to be studied normally operates at an equilibrium where all variables stay at approximately the same values, and second, that the constraint model can be written as a set of equations where an output value is some simple function of the input values.

This latter assumption is what allows DOC to use a simple constraint propagation algorithm to compute diagnoses. The contributions of DOC are in the ability to merge quantitative data from measurements into a qualitative model, and a theory of diagnosis based on *partial explanation models*. The latter constructs are the converse of conflict sets in that they are disjunctions of possible parameter value changes that could affect the observed faulty sensor values in the given manner. These partial explanations are apparently very useful in the steady-state systems in which DOC was tested, but it is not clear if they are applicable in more general diagnostic contexts.

Finally, a system that is currently being built as part of the European Esprit Project, TIGER, to diagnose faults in gas turbines, is the CA-EN algorithm [Bousson, Zimmer, & Travé-Massuyès, 1994]. This system works like a hybrid between MIMIC and GDE. As in MIMIC, dependency tracing is carried out on a causal model after the fault has been discovered. The difference here is that conflicts are generated by looking at the possible combinations of components that could have resulted in the particular variable having the given value. This approach avoids the problems with acausal predictions, discussed in Chapter 2, by ensuring that predictive inferences are propagated only in a causal manner. QDOCS, on the other hand, allows acausal predictions, and simply uses the dependencies from the actual computation of the predictions. The causal engine that CA-EN uses requires models where all predictions can be made in a causal manner, while QDOCS can use general constraint-based models with acausal predictions.

6.4 Other Approaches

In this section, we will summarize some of the earlier approaches to diagnosing continuous systems and show how they differ from QDOCS.

MIDAS [Oyeleye, Finch, & Kramer, 1990] was a model-based diagnosis system applied to dynamic systems that used a particular modeling scheme called a *signed directed graph* or SDG. An SDG is a pre-computed influence diagram on the system that has information on how particular variables affect others. When a fault was detected, MIDAS would track through the SDG looking for parameters that could have caused the observed result. An SDG, while being a computationally efficient diagnostic tool, is inadequate for expressing the complex constraints of a QSIM model. The diagnostic power of a MIDAS model is thus correspondingly reduced when compared with a model in QDOCS.

One system that was used for the diagnosis of analog circuits was CATS [Dague, Jehl, Deves, Luciani, & Taillibert, 1991]. CATS relies on a very specialized propagation mechanism called *interval propagation* that allows for uncertainties in numerical values to get propagated through constraints. Like some of the other systems mentioned, CATS relies on being able to propagate these values through constraints without having to resort to more general constraint satisfaction algorithms like QDOCS.

Finally, another model-based system that has found practical use in the diagnosis of aircraft engines is DRAPHYS [Abbott, 1988]. However, this system while using locality information from the point of fault discovery to isolate sets of components where the faults may have occurred, uses a very basic generate-and-test algorithm to actually compute the diagnoses. It therefore does not have the more efficient conflict-based diagnosis strategy of QDOCS.

Chapter 7

Future Work

In the preceding chapters, we have described our algorithms, introduced our implemented system, QDOCS, described the results of some of our experiments, and related our algorithms to some of the other approaches to the diagnosis problem. In this chapter, we will propose a number of different extensions of our research that we believe show promise and would allow the QDOCS approach to be used in actual industrial applications.

These extensions can be divided into three separate categories. In the first section, we will describe improvements to our algorithms that would allow the system to become more accurate and efficient. The second section will describe implementation improvements that we believe would improve the efficiency of QDOCS. Finally, the third section will introduce some ideas about where this research may go in the future and what it would take to get QDOCS situated in actual applications.

7.1 Algorithm Improvements

7.1.1 Across-time Verification

Currently, as we pointed out in Chapter 4, the algorithm for across-time verification, as implemented in QDOCS, is unable to produce useful conflict sets from failures to verify hypotheses across time. The methods we propose in that chapter are not robust and cost too much in computation time for very little gain in terms of useful conflict sets. This is why QDOCS resorts to the practice of returning the entire hypothesis as the conflict set when it fails to verify a behavior across time.

In most cases, however, there actually is enough information in the model and the states to obtain a better accounting of the failure. Across-time failures are usually the result of some particular variable failing to have a consistent value between states. An improved verifier might work by attempting to explain the failure in terms of specific variables that

have no consistent sequences of values. The verifier could then find dependencies for that particular variable in each of the states in which the variable failed to have a successor and explain the failure as being due to the union of these.

Isolating the particular variable or variables that caused the discontinuity is, however, a difficult task in itself, and this is open to further study. Some kind of exhaustive accounting of variables to see which ones fail to have consistent values across time could be peformed, but that may be more computationally expensive than it is worth.

7.1.2 Caching Dependencies

We have shown that the essential ideas of our caching techniques presented in Chapter 4 are sound and we believe they need to be explored in some detail. Since we know that the Waltz-filtering algorithm computes the same values over and over again, having a cache with commonly computed values stored should be advantageous.

Since cache access times need to be low compared to the times for computing these sets, limiting the size of the cache is a key idea that will need to be explored. The currently implemented technique is to simply limit the cache to a fixed number of entries. This unfortunately means that several possibly useful cache entries may be left out, and large numbers of similar ones may provide little speedup benefit. In order to better manage the cache, we will need some sort of utility criterion to determine which entries to discard and which ones to keep. This could be based on the number of accesses to that particular cache entry, or on some kind of similarity metric that would allow the system to discard entries that make similar assumptions resulting in similar possible values as other nodes.

As we saw in Chapter 5, one of the problems with the caching scheme was that caches tended to inflate the size of dependency sets. This was because dependency sets from previously cached computations had more elements than absolutely necessary to obtain the computed reductions in possible values for variables, and these larger sets were being used to initialize the network every time. Since constraint satisfaction can only add dependencies and not remove unnecessary ones, this caused cache sizes to grow to larger sizes than if we had not used the cache to initialize these variables.

To curb this inflation of dependency set sizes, one possibility would be to simply not initialize variables using the cache during some sort of *training phase* and to cache away the values and dependencies during this period. During this phase, dependency sets in the cache would become smaller and smaller as each run of the constraint satisfier would use different sequences of constraint applications to reduce the number of variable values. This would allow for the computation of the dependency sets without the problem of dependency inflation and would make it less likely that unnecessary dependencies would enter into dependency sets during initialization of variable values.

7.2 Implementation Improvements

The implementation of QDOCS we currently have is a prototype and, as such, not every aspect of the system is fully optimized. There are a few improvements that could be made to the prototype for it to be able to perform diagnosis more efficiently.

First, the current representation of dependencies and the updating peformed on them is quite inefficient. We essentialy store entire assumption sets away as linked lists rather than converting them to a more space-saving representation. This unfortunately results in more than just space inefficiencies. Every dependency update results in recursive scans of these large list structures, and often, the creation of new list structures, resulting in time inefficiencies as well. To avoid these problems, we believe a much more efficient representation might be some sort of array with simple iterative operations to perform updates.

The caching scheme could also be improved using similar modifications. As we saw in Chapter 5, the caching scheme suffers from the utility problem in that cache update and access times may outweigh the benefits. Using some kind of bit-array encoding for assumptions in that scenario could also improve access and update times of the cache which would in turn help to speed up QDOCS and show some real benefits over using it without caching.

Finally, since the across-time verification is the most computationally expensive part of QDOCS, there must be improvements made in this segment of the code. Possible improvements include search heuristics that would suggest which successor states for any given state are more likely to lead to successors that match the next set of sensor values, as opposed to a blind depth-first search that we carry out now. These heuristics might possibly include an analysis of the trends in the values of variables and a comparison with values in completions of states matching the next set of sensor values.

7.3 Incorporation into Industrial Monitoring Systems

The real test of the usefulness of the techniques we have presented here must take place in the real world. Before a QDOCS-like system actually gets situated in an industrial application, however, several problems will need to be solved. There are three broad areas in which improvements will have to be made. The first is the problem of actually automating the modeling of realistic problems, the second is that of setting up a useful monitoring system and integrating it with a QDOCS-like diagnosis system and the third is improving the diagnosis system so that it can make use of the kinds of information that an actual monitoring system can provide.

There has been significant progress in recent years on the first front. In particular, the Qualitative Process Compiler [Farquhar, 1993] which composes qualitative models from

model fragments and a description of a scenario, holds promise as a technique that will allow large models to be built automatically from simpler ones stored in model libraries. The question of how exactly this technique could be used to build models in real-world diagnostic applications is an issue that is open to further research.

The second area that must be further explored in order to make QDOCS a viable on-site diagnosis system is to merge it with a monitoring system. This is likely to not be a very difficult task, however, since QDOCS was designed with monitoring systems in mind. MIMIC and DIAMON, two of the most prominent qualitative model-based monitoring systems that have been published in the literature, both use a diagnostic module that takes an anomalous behavior of the system and proposes candidate hypotheses to be consistent with this anomaly. This is essentially what QDOCS does.

The final area in which QDOCS will have to improve is in actually being able to use the kinds of information that these monitoring systems can provide the diagnosis system. For the purposes of this prototype we have used purely qualitative models and thus we assume purely qualitative inputs from the sensors. However, actual sensor values received by monitoring systems are usually numerical values. The numerical information from the sensors could help a monitoring and diagnosis system choose between different qualitative states and determine more precisely the numerical values of landmark values for variables that may initially be unknown or known only very imprecisely.

Because of this, a number of systems have been built [Kuipers & Berleant, 1988; Kay & Kuipers, 1993] that use QSIM as the main simulation engine but allow for more of an integration of numerical values into the simulation to make the behaviors more precise. A diagnosis system using a semi-quantitative model such as those used by these simulation systems would need to, in addition to propagating dependencies during constraint application, also propagate them during interval updating. The techniques for doing this and the effects in being able to improve diagnoses is an issue that must be addressed in future research.

Chapter 8

Conclusions

As systems get more and more complex, automatic monitoring and diagnosis systems will become more important for keeping track of what these systems are doing in real time, and to determine in what ways they may be broken. Since computational complexity, and the lack of the right kinds of available knowledge, preclude the option of building precise numerical models, qualitative modeling for diagnosis will become especially important.

Our research has made important contributions to the field of model-based diagnosis in continuous dynamic systems using qualitative modeling. Most of the previous work on model-based diagnosis has been on static systems like combinational circuits, while most of the work on the diagnosis of continuous dynamic systems has suffered from various limitations in its domain of applicability.

We have designed and implemented an algorithm for the diagnosis of behaviors of dynamic systems which is able to diagnose multiple faults in the context of behavioral modes. It adds a technique for propagating dependencies while using a general constraint satisfaction algorithm, and also verifies the correctness of hypotheses by checking that the entire behavior is a valid simulation of the underlying model corresponding to the hypothesis.

We have demonstrated the usefulness of the algorithm in diagnosing simulated behaviors on two realistic systems - the reaction control system of the space shuttle, and a level-controlled reaction tank. We have shown that QDOCS outperforms a simpler generate-and-test algorithm in these domains, and, through ablation tests, we have demonstrated the utility of all the components of our system.

In comparisons with related systems, we have shown that QDOCS is unique in that it uses a general constraint language for modeling dynamic systems, and it is able to use a constraint satisfaction algorithm to test the consistency of hypotheses and to compute conflict sets and thus it is not limited to systems for which constraint propagation is not blocked. We have also shown that unlike some of these systems, it avoids making simplifying

assumptions that limit the range of hypotheses to single faults alone.

Finally, we have suggested future research directions which will improve the accuracy, efficiency, and applicability of our techniques. Our research leads us to believe that online diagnosis of complex dynamic systems is, indeed, viable, and holds great promise for the future.

Appendix A

The Reaction Control System

A.1 QDOCS Model of the RCS

Given below is the QSIM model, followed by the `defComponents` and `defOperatingRegions` descriptions of the RCS. The QSIM model is a modified and extended version of Kay's model [1992]. All variables that are duplicated in the two propellant subsystems have identical names in the two subsystems except that they end in either a 0 or 1 depending on the subsystem. The propellant is referred to as *fuel* in the variables, even though it may refer to the oxidizer.

```
(define-QDE rcs-double-model
   (text "RCS helium and fuel tanks")
   (quantity-spaces
      (AmtHe0       (0 AmtHeMin AmtHeLow AmtHePreg AmtHeSreg
                     AmtHeInit))
                    ; Amt of Helium in He Tank
      (PHe0         (0 PHeMin PHeLow PHePreg PHeSreg PHeInit))
                    ; Pressure in He tank
      (dAmtHe0      (minf 0 inf))
                    ; Change in amt of He
      (PHeSensed0   (0 PHeMin PHeLow PHePreg PHeSreg PHeInit))
                    ; Sensed pressure in He tank
      (He->UllFlow0 (0 inf))       ; Flow from He tank to He line
      (HeLeakflow0  (0 inf))       ; Helium leak flow
      (HeOutFlow0   (0 inf))       ; Total flow out of the He tank
      (PSReg0       (0 PUllMin PullPreg PullSreg PUllMax))
                    ; pressure past the secondary reg
      (AmtUll0      (0 AmtUllInit AmtUllNom AmtUllHigh AmtUllMax))
                    ; Amt ullage in prop Tank
```

```
(nrt[AmtUll]0 (0 nRTInit nRTNom nRTHigh nRTMax))
              ; Amt in correct units
(PUll0        (0 PUllMin PUllPreg  PUllSreg PUllMax))
              ; Pressure in Propellant tank
(PUllSensed0  (0 PUllMin PUllPreg  PUllSreg PUllMax))
              ; Sensed version of PUll0
(AmtFuel0     (0 AmtFNom AmtFInit AmtFMax))
              ; Amount of fuel in prop tank
(VolFuel0     (0 VFLow VFNom VFInit VolTotal))
              ; Volume of fuel
(VolFuelSensed0 (0 VFLow VFNom VFInit VolTotal))
              ; Fuel gauge reading
(VolUll0      (0 VUInit VUNom  VUHigh VolTotal))
              ; Volume of Ullage
(VolTotal0    (0 VolTotal))  ; Total vol of prop tank
(DenFuel0     (0 DFPreg DFMax))
              ; fuel density (a function of pressure)
(dAmtFuel0    (minf 0 inf))  ; change in fuel in tank
(Fuel->ManFlow0  (minf 0 inf))
              ; flow proptank -> manifold
(PDiffFuel0   (minf 0 inf))  ; PUll-PMan
(TIValve0     (0 Max))
              ; combined tank & man vlve conductance
(AmtMan0      (0 AmtPreg AmtMax))
              ; Amount of fuel in manifold
(dAmtMan0     (minf 0 inf))  ; change in manifold quantity
(PMan0        (0 PManPreg PManMax))
              ; Manifold pressure
(PManSensed0  (0 PManPreg PManMax))
              ; Manifold pressure sensed
(Man->ThFlow0 (minf 0 inf))  ; flow Manifold -> Thruster
(PDiffMan0    (minf 0 PD* inf))
              ; PMan-PVac
(DenFuelMan0  (0 inf))       ; Density of fuel in manifold
(DPFuel0      (0 inf))       ; Density*PDiffMan
(f[DPFuel]0   (0 inf))       ; CDa*f[DPFuel] = flow in lbs/sec
(PVac0        (0 inf))       ; Pressure of space
(LineLeakS0   (0 max))       ; line leak condctnce
(Man->Leakflow0  (0 inf))    ; total leak flow (From man and line)
(ManLeak0        (0 inf))    ; manifold leak flow
(Man->LineLeak0  (0 inf))    ; leak flow from line
(ManLeakS0       (0 max))    ; manifold leak conductance
(PDiffManAndVac0 (0 inf))    ; pressure diff between man and vac
```

```
(ManOutflow0     (minf 0 inf))
              ; total flow out of manifold
(MManOutflow0    (minf 0 inf))
              ; -ManOutflow
(TotalHe0     (0 TotalHeNorm))
              ; Total Helium - add landmarks
(dTotHe0      (minf 0))
              ; Derivative of Total Helium

;; The other propellant tank...
(AmtHe1       (0 AmtHeMin AmtHeLow AmtHePreg AmtHeSreg
               AmtHeInit))
              ; Amt of Helium in He Tank
(PHe1         (0 PHeMin PHeLow PHePreg PHeSreg PHeInit))
              ; Pressure in He tank
(dAmtHe1      (minf 0 inf))
              ; Change in amt of He
(PHeSensed1   (0 PHeMin PHeLow PHePreg PHeSreg PHeInit))
              ; Sensed pressure in He tank
(He->UllFlow1 (0 inf))       ; Flow from He tank to He line
(HeLeakflow1  (0 inf))       ; Helium leak flow
(HeOutFlow1   (0 inf))       ; Total flow out of the He tank
(PSReg1       (0 PUllMin PullPreg PullSreg PUllMax))
              ; pressure past the secondary reg
(AmtUll1      (0 AmtUllInit AmtUllNom AmtUllHigh AmtUllMax))
              ; Amt ullage in prop Tank
(nrt[AmtUll]1 (0 nRTInit nRTNom nRTHigh nRTMax))
              ; Amt in correct units
(PUll1        (0 PUllMin PUllPreg  PUllSreg PUllMax))
              ; Pressure in Propellant tank
(PUllSensed1  (0 PUllMin PUllPreg  PUllSreg PUllMax))
              ; Sensed version of PUll1
(AmtFuel1     (0 AmtFNom AmtFInit AmtFMax))
              ; Amount of fuel in prop tank
(VolFuel1     (0 VFLow VFNom VFInit VolTotal))
              ; Volume of fuel
(VolFuelSensed1 (0 VFLow VFNom VFInit VolTotal))
              ; Fuel gauge reading
(VolUll1      (0 VUInit VUNom  VUHigh VolTotal))
              ; Volume of Ullage
(VolTotal1    (0 VolTotal))  ; Total vol of prop tank
(DenFuel1     (0 DFPreg DFMax))
              ; fuel density (a function of pressure)
```

```
   (dAmtFuel1    (minf 0 inf))  ; change in fuel in tank
   (Fuel->ManFlow1  (minf 0 inf))
                 ; flow proptank -> manifold
   (PDiffFuel1   (minf 0 inf))  ; PUll-PMan
   (TIValve1     (0 Max))
                 ; combined tank & man vlve conductance
   (AmtMan1      (0 AmtPreg AmtMax))
                 ; Amount of fuel in manifold
   (dAmtMan1     (minf 0 inf))  ; change in manifold quantity
   (PMan1        (0 PManPreg PManMax))
                 ; Manifold pressure
   (PManSensed1  (0 PManPreg PManMax))
                 ; Manifold pressure sensed
   (Man->ThFlow1 (minf 0 inf))  ; flow Manifold -> Thruster
   (PDiffMan1    (minf 0 PD* inf))
                 ; PMan-PVac
   (DenFuelMan1  (0 inf))       ; Density of fuel in manifold
   (DPFuel1      (0 inf))       ; Density*PDiffMan
   (f[DPFuel]1   (0 inf))       ; CDa*f[DPFuel] = flow in lbs/sec
   (PVac1        (0 inf))       ; Pressure of space
   (LineLeakS1   (0 max))       ; line leak condctnce
   (Man->Leakflow1  (0 inf))    ; total leak flow (From man and line)
   (ManLeak1        (0 inf))    ; manifold leak flow
   (Man->LineLeak1  (0 inf))    ; leak flow from line
   (ManLeakS1       (0 max))    ; manifold leak conductance
   (PDiffManAndVac1 (0 inf))    ; pressure diff between man and vac
   (ManOutflow1     (minf 0 inf))
                 ; total flow out of manifold
   (MManOutflow1    (minf 0 inf))
                 ; -ManOutflow
   (TotalHe1     (0 TotalHeNorm))
                 ; Total Helium - add landmarks
   (dTotHe1      (minf 0))
                 ; Derivative of Total Helium

   (Thruster     (0 Max))       ; Mode of the Thruster - common between
                                  propellant tanks

   )
(discrete-variables
   (preg-mode0 (normal stuck-open stuck-closed unknown))
               ; Mode of the primary regulator
   (sreg-mode0 (normal stuck-open stuck-closed unknown))
```

```
            ; Mode of the secondary regulator
(preg-region0 (ideal pipe))
            ; Operating region of primary regulator
(sreg-region0 (ideal pipe))
            ; Operating region of secondary regulator
(helium-system-mode0 (normal leaking unknown))
            ; Mode of helium tank and line
(he-tank-p-sensor-mode0 (normal stuck-at-0 unknown))
            ; Mode of Helium tank pressure sensor
(ullage-pressure-sensor-mode0 (normal stuck-at-0 unknown))
            ; Mode of Ullage pressure sensor
(fuel-volume-sensor-mode0 (normal stuck-at-0 unknown))
            ; Mode of propellant volume sensor
(manifold-pressure-sensor-mode0 (normal stuck-at-0 unknown))
            ; Mode of manifold pressure sensor
(tankvalve0 (open closed))
            ; State of tank valve
(manvalve0 (open closed))
            ; State of manifold valve
(manvalve-mode0 (normal stuck-open stuck-closed unknown))
            ; Mode of manifold valve
(tankvalve-mode0 (normal stuck-open stuck-closed unknown))
            ; Mode of tank valve
(manifold-mode0 (normal leaking unknown))
            ; Mode of manifold
(fuelline-mode0 (normal leaking unknown))
            ; Mode of the fuel line
;; The other propellant tank...
(preg-mode1 (normal stuck-open stuck-closed unknown))
(sreg-mode1 (normal stuck-open stuck-closed unknown))
(preg-region1 (ideal pipe))
(sreg-region1 (ideal pipe))
(helium-system-mode1 (normal leaking unknown))
(he-tank-p-sensor-mode1 (normal stuck-at-0 unknown))
(ullage-pressure-sensor-mode1 (normal stuck-at-0 unknown))
(fuel-volume-sensor-mode1 (normal stuck-at-0 unknown))
(manifold-pressure-sensor-mode1 (normal stuck-at-0 unknown))
(tankvalve1 (open closed))
(manvalve1 (open closed))
(manvalve-mode1 (normal stuck-open stuck-closed unknown))
(tankvalve-mode1 (normal stuck-open stuck-closed unknown))
(manifold-mode1 (normal leaking unknown))
(fuelline-mode1 (normal leaking unknown))
```

```
   (thruster-mode (normal stuck-open stuck-closed unknown))
               ; Mode of the thruster
   )

(constraints

 ;; Helium tank
 ((d/dt AmtHe0 dAmtHe0))
 ((minus dAmtHe0 HeOutflow0))
 ((M+ AmtHe0 PHe0)   (0 0)  (AmtHeLow PHeLow) (AmtHeMin PHeMin)
  (AmtHePreg PHePreg) (AmtHeSreg PHeSreg) (AmtHeInit PHeInit))

 ;; Helium Leaks
 (mode (helium-system-mode0 normal)
       ((zero-std HeLeakFlow0)))
 ((add HeLeakFlow0 He->UllFlow0 HeOutFlow0) (0 0 0)
  (inf inf inf))
 (mode (helium-system-mode0 leaking)
       ((M+ PHe0 HeLeakFlow0) (0 0)))

 ;; Integrity constraints on pressure regulator regions.
 (mode (and (sreg-mode0 normal) (sreg-region0 ideal))
       ((greater-than-or-equal PHe0 PHeSreg)))
 (mode (and (preg-mode0 normal) (preg-region0 ideal))
       ((greater-than-or-equal PSReg0 PUllPreg)))
 (mode (and (sreg-mode0 normal) (sreg-region0 pipe))
       ((less-than-or-equal PHe0 PHeSreg)))
 (mode (and (preg-mode0 normal) (preg-region0 pipe))
       ((less-than-or-equal PSReg0 PUllPreg)))

 ;; Effects of Pressure Regulators
 (mode (and (sreg-mode0 normal) (sreg-region0 ideal))
       ((constant PSReg0 PUllSreg)))
 (mode (or (and (sreg-mode0 normal) (sreg-region0 pipe))
           (sreg-mode0 stuck-open))
       ((equal PSReg0 PHe0) (0 0) (PUllPreg PHePreg)
        (PUllSreg PHeSreg)))
 (mode (and (preg-mode0 normal) (preg-region0 ideal))
       ((constant PUll0 PUllPreg)))
 (mode (or (and (preg-mode0 normal) (preg-region0 pipe))
           (preg-mode0 stuck-open))
       ((equal PUll0 PSReg0) (0 0) (PullMin PUllMin) (PUllPreg PUllPreg)
```

```
       (PUllSreg PUllSreg) (PUllMax PUllMax)))


;; Flow from HeTank to Ullage tank
(mode (and (or (preg-mode0 normal) (preg-mode0 stuck-open))
           (or (sreg-mode0 normal) (sreg-mode0 stuck-open)))
      ((equal Fuel->Manflow0 He->UllFlow0) (0 0) (inf inf)))
(mode (or (preg-mode0 stuck-closed) (sreg-mode0 stuck-closed))
      ((zero-std He->UllFlow0)))
((d/dt AmtUll0 He->UllFlow0))

;; Propellant tank - propellant flow
((d/dt AmtFuel0 dAmtFuel0))
((minus Fuel->Manflow0 dAmtFuel0))

;; Density, amount and volume of propellant
((M+ PUll0 DenFuel0) (0 0) (PUllPreg DFPreg) (PUllMax DFMax))
((mult DenFuel0 VolFuel0 AmtFuel0) (DFPreg VFInit AmtFInit)
 (DFPreg VFNom AmtFNom))
((M+ AmtUll0 nRT[AmtUll]0) (0 0) (AmtUllInit nRTInit)
 (AmtUllNom nRTNom) (AmtUllHigh nRTHigh)
 (AmtUllMax nRTMax))
((mult PUll0 VolUll0 nRT[AmtUll]0) (0 0 0) (PUllPReg VUInit nRTInit)
 (PUllPReg VUNom nRTNom)
 (PUllPReg VUHigh nRTHigh)
 (PUllMax VolTotal nRTMax))
((add VolFuel0 VolUll0 VolTotal0) (VolTotal 0 VolTotal)
 (0 VolTotal VolTotal)
 (VFNom VUNom VolTotal)
 (VFLow VUHigh VolTotal)
 (VFInit VUInit VolTotal))

;; Total Helium in the system normally remains constant.
((add AmtUll0 AmtHe0 TotalHe0) (0 0 0)
 (AmtUllNom AmtHeInit TotalHeNorm)
 (AmtUllHigh AmtHeLow TotalHeNorm))
((minus dTotHe0 HeLeakflow0))
((d/dt TotalHe0 dTotHe0))
(mode (helium-system-mode0 normal)
      ((constant TotalHe0 TotalHeNorm)))
((constant VolTotal0 VolTotal))

;; Sensor modes
```

```
(mode (he-tank-p-sensor-mode0 normal)
      ((equal PHeSensed0 PHe0) (0 0) (PHeMin PHeMin) (PHeLow PHeLow)
       (PHePreg PHePreg) (PHeSreg PHeSreg) (PHeInit PHeInit)))
(mode (ullage-pressure-sensor-mode0 normal)
      ((equal PUllSensed0 PUll0) (0 0) (PUllMin PUllMin)
       (PUllPreg PUllPreg) (PUllSreg PUllSreg) (PUllMax PUllMax)))
(mode (fuel-volume-sensor-mode0 normal)
      ((equal VolFuelSensed0 VolFuel0) (0 0) (VFLow VFLow)
       (VFNom VFNom) (VFInit VFInit) (VolTotal VolTotal)))
(mode (he-tank-p-sensor-mode0 stuck-at-0)
      ((zero-std PHeSensed0)))
(mode (ullage-pressure-sensor-mode0 stuck-at-0)
      ((zero-std PUllSensed0)))
(mode (fuel-volume-sensor-mode0 stuck-at-0)
      ((zero-std VolFuelSensed0)))
(mode (manifold-pressure-sensor-mode0 normal)
      ((equal PMan0 PManSensed0)
       (0 0) (PManPreg PManPreg) (PManMax PManMax)))
(mode (manifold-pressure-sensor-mode0 stuck-at-0)
      ((zero-std PManSensed0)))

;; Manifold pressures and flows
((add PDiffFuel0 PMan0 PUll0) (0 PManPreg PUllPreg))
((mult PDiffFuel0 TIValve0 Fuel->ManFlow0))
(mode (and (or (and (tankvalve-mode0 normal) (TankValve0 open))
               (tankvalve-mode0 stuck-open))
           (or (and (manvalve-mode0 normal) (ManValve0 open))
               (manvalve-mode0 stuck-open)))
      ((constant TIValve0 max)))
(mode (or (and (tankvalve-mode0 normal)
               (TankValve0 closed))
          (and (manvalve-mode0 normal)
               (ManValve0 closed))
          (tankvalve-mode0 stuck-closed))
      ((zero-std TIValve0)))
((constant tankvalve0))
((constant manvalve0))
((constant PVac0 0))
((d/dt AmtMan0 dAmtMan0))
((M+ AmtMan0 PMan0) (0 0) (AmtPreg PManPreg) (AmtMax PManMax))
((add PDiffMan0 PVac0 PMan0) (PD* 0 PManPreg))
((mult PDiffMan0 DenFuelMan0 DPFuel0))
((M+ PMan0 DenFuelMan0) (0 0) (PManMax inf))
```

```
((M+ DPFuel0 f[DPFuel]0) (0 0) (inf inf))

;; Manifold -> Thruster flow
((mult Thruster f[DPFuel]0 Man->ThFlow0))
((add Fuel->ManFlow0 MManOutflow0 dAmtMan0))
((W+ PDiffMan0 DPFuel0))

;; Leaks in the Manifold
((add Man->Leakflow0 Man->ThFlow0 ManOutflow0))
((minus MManOutflow0 ManOutflow0))
((add ManLeak0 Man->LineLeak0 Man->Leakflow0))
((add PDiffManAndVac0 PVac0 PMan0))
((mult ManLeakS0 PDiffManAndVac0 ManLeak0))
(mode (fuelline-mode0 leaking)
      ((mult LineLeaks0 PDiffManAndVac0 Man->LineLeak0)))
(mode (fuelline-mode0 normal)
      ((zero-std Man->LineLeak0)))
(mode (manifold-mode0 leaking)
      ((constant ManLeakS0)))
(mode (manifold-mode0 normal)
      ((zero-std ManLeakS0)))
(mode (fuelline-mode0 leaking)
      ((constant lineleaks0)))
(mode (fuelline-mode0 normal)
      ((zero-std lineleaks0)))

;; Thruster modes
(mode (thruster-mode normal)
      ((constant thruster max)))
(mode (thruster-mode stuck-closed)
      ((constant thruster 0)))

;; The other propellant tank...

;; Helium tank
((d/dt AmtHe1 dAmtHe1))
((minus dAmtHe1 HeOutflow1))
((M+ AmtHe1 PHe1)   (0 0)  (AmtHeLow PHeLow) (AmtHeMin PHeMin)
 (AmtHePreg PHePreg) (AmtHeSreg PHeSreg) (AmtHeInit PHeInit))

;; Helium Leaks
(mode (helium-system-mode1 normal)
      ((zero-std HeLeakFlow1)))
```

```
((add HeLeakFlow1 He->UllFlow1 HeOutFlow1) (0 0 0)
 (inf inf inf))
(mode (helium-system-mode1 leaking)
      ((M+ PHe1 HeLeakFlow1) (0 0)))

;; Integrity constraints on pressure regulator regions.
(mode (and (sreg-mode1 normal) (sreg-region1 ideal))
      ((greater-than-or-equal PHe1 PHeSreg)))
(mode (and (preg-mode1 normal) (preg-region1 ideal))
      ((greater-than-or-equal PSReg1 PUllPreg)))
(mode (and (sreg-mode1 normal) (sreg-region1 pipe))
      ((less-than-or-equal PHe1 PHeSreg)))
(mode (and (preg-mode1 normal) (preg-region1 pipe))
      ((less-than-or-equal PSReg1 PUllPreg)))

;; Effects of Pressure Regulators
(mode (and (sreg-mode1 normal) (sreg-region1 ideal))
      ((constant PSReg1 PUllSreg)))
(mode (or (and (sreg-mode1 normal) (sreg-region1 pipe))
          (sreg-mode1 stuck-open))
      ((equal PSReg1 PHe1) (0 0) (PUllPreg PHePreg)
       (PUllSreg PHeSreg)))
(mode (and (preg-mode1 normal) (preg-region1 ideal))
      ((constant PUll1 PUllPreg)))
(mode (or (and (preg-mode1 normal) (preg-region1 pipe))
          (preg-mode1 stuck-open))
      ((equal PUll1 PSReg1) (0 0) (PullMin PUllMin) (PUllPreg PUllPreg)
       (PUllSreg PUllSreg) (PUllMax PUllMax)))


;; Flow from HeTank to Ullage tank
(mode (and (or (preg-mode1 normal) (preg-mode1 stuck-open))
           (or (sreg-mode1 normal) (sreg-mode1 stuck-open)))
      ((equal Fuel->Manflow1 He->UllFlow1) (0 0) (inf inf)))
(mode (or (preg-mode1 stuck-closed) (sreg-mode1 stuck-closed))
      ((zero-std He->UllFlow1)))
((d/dt AmtUll1 He->UllFlow1))

;; Propellant tank - propellant flow
((d/dt AmtFuel1 dAmtFuel1))
((minus Fuel->Manflow1 dAmtFuel1))

;; Density, amount and volume of propellant
```

```
((M+ PUll1 DenFuel1) (0 0) (PUllPreg DFPreg) (PUllMax DFMax))
((mult DenFuel1 VolFuel1 AmtFuel1) (DFPreg VFInit AmtFInit)
 (DFPreg VFNom AmtFNom))
((M+ AmtUll1 nRT[AmtUll]1) (0 0) (AmtUllInit nRTInit)
 (AmtUllNom nRTNom) (AmtUllHigh nRTHigh)
 (AmtUllMax nRTMax))
((mult PUll1 VolUll1 nRT[AmtUll]1) (0 0 0) (PUllPReg VUInit nRTInit)
 (PUllPReg VUNom nRTNom)
 (PUllPReg VUHigh nRTHigh)
 (PUllMax VolTotal nRTMax))
((add VolFuel1 VolUll1 VolTotal1) (VolTotal 0 VolTotal)
 (0 VolTotal VolTotal)
 (VFNom VUNom VolTotal)
 (VFLow VUHigh VolTotal)
 (VFInit VUInit VolTotal))

;; Total Helium in the system normally remains constant.
((add AmtUll1 AmtHe1 TotalHe1) (0 0 0)
 (AmtUllNom AmtHeInit TotalHeNorm)
 (AmtUllHigh AmtHeLow TotalHeNorm))
((minus dTotHe1 HeLeakflow1))
((d/dt TotalHe1 dTotHe1))
(mode (helium-system-mode1 normal)
      ((constant TotalHe1 TotalHeNorm)))
((constant VolTotal1 VolTotal))

;; Sensor modes
(mode (he-tank-p-sensor-mode1 normal)
      ((equal PHeSensed1 PHe1) (0 0) (PHeMin PHeMin) (PHeLow PHeLow)
       (PHePreg PHePreg) (PHeSreg PHeSreg) (PHeInit PHeInit)))
(mode (ullage-pressure-sensor-mode1 normal)
      ((equal PUllSensed1 PUll1) (0 0) (PUllMin PUllMin)
       (PUllPreg PUllPreg) (PUllSreg PUllSreg) (PUllMax PUllMax)))
(mode (fuel-volume-sensor-mode1 normal)
      ((equal VolFuelSensed1 VolFuel1) (0 0) (VFLow VFLow)
       (VFNom VFNom) (VFInit VFInit) (VolTotal VolTotal)))
(mode (he-tank-p-sensor-mode1 stuck-at-0)
      ((zero-std PHeSensed1)))
(mode (ullage-pressure-sensor-mode1 stuck-at-0)
      ((zero-std PUllSensed1)))
(mode (fuel-volume-sensor-mode1 stuck-at-0)
      ((zero-std VolFuelSensed1)))
(mode (manifold-pressure-sensor-mode1 normal)
```

```
      ((equal PMan1 PManSensed1)
       (0 0) (PManPreg PManPreg) (PManMax PManMax)))
(mode (manifold-pressure-sensor-mode1 stuck-at-0)
      ((zero-std PManSensed1)))

;; Manifold pressures and flows
((add PDiffFuel1 PMan1 PUll1) (0 PManPreg PUllPreg))
((mult PDiffFuel1 TIValve1 Fuel->ManFlow1))
(mode (and (or (and (tankvalve-mode1 normal) (TankValve1 open))
               (tankvalve-mode1 stuck-open))
           (or (and (manvalve-mode1 normal) (ManValve1 open))
               (manvalve-mode1 stuck-open)))
      ((constant TIValve1 max)))
(mode (or (and (tankvalve-mode1 normal)
               (TankValve1 closed))
          (and (manvalve-mode1 normal)
               (ManValve1 closed))
          (tankvalve-mode1 stuck-closed))
      ((zero-std TIValve1)))
((constant tankvalve1))
((constant manvalve1))
((constant PVac1 0))
((d/dt AmtMan1 dAmtMan1))
((M+ AmtMan1 PMan1) (0 0) (AmtPreg PManPreg) (AmtMax PManMax))
((add PDiffMan1 PVac1 PMan1) (PD* 0 PManPreg))
((mult PDiffMan1 DenFuelMan1 DPFuel1))
((M+ PMan1 DenFuelMan1) (0 0) (PManMax inf))
((M+ DPFuel1 f[DPFuel]1) (0 0) (inf inf))

;; Manifold -> Thruster flow
((mult Thruster f[DPFuel]1 Man->ThFlow1))
((add Fuel->ManFlow1 MManOutflow1 dAmtMan1))
((W+ PDiffMan1 DPFuel1))

;; Leaks in the Manifold
((add Man->Leakflow1 Man->ThFlow1 ManOutflow1))
((minus MManOutflow1 ManOutflow1))
((add ManLeak1 Man->LineLeak1 Man->Leakflow1))
((add PDiffManAndVac1 PVac1 PMan1))
((mult ManLeakS1 PDiffManAndVac1 ManLeak1))
(mode (fuelline-mode1 leaking)
      ((mult LineLeaks1 PDiffManAndVac1 Man->LineLeak1)))
(mode (fuelline-mode1 normal)
```

```
        ((zero-std Man->LineLeak1)))
    (mode (manifold-mode1 leaking)
        ((constant ManLeakS1)))
    (mode (manifold-mode1 normal)
        ((zero-std ManLeakS1)))
    (mode (fuelline-mode1 leaking)
        ((constant lineleaks1)))
    (mode (fuelline-mode1 normal)
        ((zero-std lineleaks1)))

    )

   ;; Region Transitions
   (transitions ((and (sreg-mode0 (normal std)) (sreg-region0 (ideal std))
                      (PHe0 (PHeSreg nil)))
                 -> sreg-pipe0)
                ((and (preg-mode0 (normal std)) (preg-region0 (ideal std))
                      (PSReg0 (PUllPreg nil)))
                 -> preg-pipe0)
                ((and (sreg-mode1 (normal std)) (sreg-region1 (ideal std))
                      (PHe1 (PHeSreg nil)))
                 -> sreg-pipe1)
                ((and (preg-mode1 (normal std)) (preg-region1 (ideal std))
                      (PSReg1 (PUllPreg nil)))
                 -> preg-pipe1)
                )
    ))


;;; These are the actual region transition functions.

(defun sreg-pipe0 (state)
  ;;(format t "~%-> sreg-pipe0")
  (create-transition-state
   :from-state state
   :to-qde rcs-double-model
   :assert '((sreg-region0 (pipe std)))
   :inherit-qmag #'all-except-helium-flow
   :inherit-qdir nil))


(defun preg-pipe0 (state)
  ;;(format t "~%-> preg-pipe0")
```

```
  (create-transition-state
   :from-state state
   :to-qde rcs-double-model
   :assert '((preg-region0 (pipe std)))
   :inherit-qmag #'all-except-helium-flow
   :inherit-qdir nil))

(defun sreg-pipe1 (state)
  ;;(format t "~%-> sreg-pipe1")
  (create-transition-state
   :from-state state
   :to-qde rcs-double-model
   :assert '((sreg-region1 (pipe std)))
   :inherit-qmag #'all-except-helium-flow
   :inherit-qdir nil))


(defun preg-pipe1 (state)
  ;;(format t "~%-> preg-pipe1")
  (create-transition-state
   :from-state state
   :to-qde rcs-double-model
   :assert '((preg-region1 (pipe std)))
   :inherit-qmag #'all-except-helium-flow
   :inherit-qdir nil))

;;; A function used by the region transition functions
(defun all-except-helium-flow (varname)
  (not (member varname '(He->UllFlow0 HeOutFlow0 He->UllFlow1
                         HeOutFlow1))))



;;; The components specification for the RCS system.
(defComponents rcs-double-model
  (primary-regulator0 preg-mode0 (normal . 0.98) (stuck-open . 0.01)
                                 (stuck-closed . 0.01))
  (secondary-regulator0 sreg-mode0 (normal . 0.98) (stuck-open . 0.01)
                                   (stuck-closed . 0.01))
  (helium-system0 helium-system-mode0 (normal . 0.98) (leaking . 0.02))
  (he-tank-sensor0 he-tank-p-sensor-mode0
                   (normal . 0.98) (stuck-at-0 . 0.02))
  (ullage-pressure-sensor0 ullage-pressure-sensor-mode0
```

```
                          (normal . 0.98) (stuck-at-0 . 0.02))
  (fuel-volume-sensor0 fuel-volume-sensor-mode0
                       (normal . 0.98) (stuck-at-0 . 0.02))
  (tankvalve0 tankvalve-mode0 (normal . 0.99) (stuck-open . 0.005)
                              (stuck-closed . 0.005))
  (manvalve0  manvalve-mode0  (normal . 0.99) (stuck-open . 0.005)
                              (stuck-closed . 0.005))
  (fuelline0  fuelline-mode0  (normal . 0.995) (leaking . 0.005))
  (manifold0  manifold-mode0  (normal . 0.995) (leaking . 0.005))
  (manifold-pressure-sensor0 manifold-pressure-sensor-mode0
                             (normal . 0.995) (stuck-at-0 . 0.005))

  (primary-regulator1 preg-mode1 (normal . 0.98) (stuck-open . 0.01)
                                 (stuck-closed . 0.01))
  (secondary-regulator1 sreg-mode1 (normal . 0.98) (stuck-open . 0.01)
                                   (stuck-closed . 0.01))
  (helium-system1 helium-system-mode1 (normal . 0.98) (leaking . 0.02))
  (he-tank-sensor1 he-tank-p-sensor-mode1
                   (normal . 0.98) (stuck-at-0 . 0.02))
  (ullage-pressure-sensor1 ullage-pressure-sensor-mode1
                   (normal . 0.98) (stuck-at-0 . 0.02))
  (fuel-volume-sensor1 fuel-volume-sensor-mode1
                   (normal . 0.98) (stuck-at-0 . 0.02))
  (tankvalve1 tankvalve-mode1 (normal . 0.99) (stuck-open . 0.005)
                              (stuck-closed . 0.005))
  (manvalve1  manvalve-mode1  (normal . 0.99) (stuck-open . 0.005)
                              (stuck-closed . 0.005))
  (fuelline1  fuelline-mode1  (normal . 0.995) (leaking . 0.005))
  (manifold1  manifold-mode1  (normal . 0.995) (leaking . 0.005))
  (manifold-pressure-sensor1 manifold-pressure-sensor-mode1
                             (normal . 0.995) (stuck-at-0 . 0.005))


  (thruster thruster-mode (normal . 0.995) (stuck-closed . 0.005))
  )



;;; The Operating regions for the RCS model.  Note that there are
;;; regions specified for every mode.  This is necessary so Qdocs
;;; can determine a value for the region variable.
(defOperatingRegions rcs-double-model
  ((preg-mode0 . normal) (preg-region0 . ideal) (preg-region0 . pipe))
```

```
((sreg-mode0 . normal) (sreg-region0 . ideal) (sreg-region0 . pipe))
((preg-mode0 . stuck-open) (preg-region0 . ideal))
((preg-mode0 . stuck-closed) (preg-region0 . ideal))
((preg-mode0 . unknown) (preg-region0 . ideal))
((sreg-mode0 . stuck-open) (sreg-region0 . ideal))
((sreg-mode0 . stuck-closed) (sreg-region0 . ideal))
((sreg-mode0 . unknown) (sreg-region0 . ideal))


((preg-mode1 . normal) (preg-region1 . ideal) (preg-region1 . pipe))
((sreg-mode1 . normal) (sreg-region1 . ideal) (sreg-region1 . pipe))
((preg-mode1 . stuck-open) (preg-region1 . ideal))
((preg-mode1 . stuck-closed) (preg-region1 . ideal))
((preg-mode1 . unknown) (preg-region1 . ideal))
((sreg-mode1 . stuck-open) (sreg-region1 . ideal))
((sreg-mode1 . stuck-closed) (sreg-region1 . ideal))
((sreg-mode1 . unknown) (sreg-region1 . ideal)))
```

Appendix B

A Sample Run of QDOCS on an RCS Diagnosis Problem

This is one of the actual problems from the test suite reported on in Chapter 5. The following is the actual sequence of observations. It is followed by an output trace. The NIL values below represent the fact that we do not know the direction of change of these sensor values. If that information is easily obtainable from the sensors, QDOCS will be able to use it.

```
(((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 (PULLSREG NIL))
  (VOLFUELSENSED0 ((VFNOM VFINIT) NIL)) (PMANSENSED0 (0 NIL))
  (PHESENSED1 (PHEINIT NIL)) (PULLSENSED1 (0 NIL))
  (VOLFUELSENSED1 (VFNOM NIL)) (PMANSENSED1 (0 NIL))
  (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL))
  (TANKVALVE1 (OPEN NIL)) (MANVALVE1 (OPEN NIL)))
 ((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
  (VOLFUELSENSED0 ((VFNOM VFINIT) NIL)) (PMANSENSED0 ((0 PMANPREG) NIL))
  (PHESENSED1 ((PHESREG PHEINIT) NIL)) (PULLSENSED1 (0 NIL))
  (VOLFUELSENSED1 ((VFLOW VFNOM) NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
  (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
  (MANVALVE1 (OPEN NIL)))
 ((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
  (VOLFUELSENSED0 ((VFNOM VFINIT) NIL)) (PMANSENSED0 (PMANPREG NIL))
  (PHESENSED1 ((PHESREG PHEINIT) NIL)) (PULLSENSED1 (0 NIL))
  (VOLFUELSENSED1 ((VFLOW VFNOM) NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
  (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
  (MANVALVE1 (OPEN NIL)))
 ((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
  (VOLFUELSENSED0 ((VFNOM VFINIT) NIL))
```

```
 (PMANSENSED0 ((PMANPREG PMANMAX) NIL))
 (PHESENSED1 ((PHESREG PHEINIT) NIL)) (PULLSENSED1 (0 NIL))
 (VOLFUELSENSED1 ((VFLOW VFNOM) NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
 (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
 (MANVALVE1 (OPEN NIL)))
((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
 (VOLFUELSENSED0 ((VFNOM VFINIT) NIL))
 (PMANSENSED0 ((PMANPREG PMANMAX) NIL))
 (PHESENSED1 (PHESREG NIL)) (PULLSENSED1 (0 NIL))
 (VOLFUELSENSED1 ((VFLOW VFNOM) NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
 (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
 (MANVALVE1 (OPEN NIL)))
((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
 (VOLFUELSENSED0 ((VFNOM VFINIT) NIL))
 (PMANSENSED0 ((PMANPREG PMANMAX) NIL))
 (PHESENSED1 ((PHEPREG PHESREG) NIL)) (PULLSENSED1 (0 NIL))
 (VOLFUELSENSED1 ((VFLOW VFNOM) NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
 (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
 (MANVALVE1 (OPEN NIL)))
((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
 (VOLFUELSENSED0 ((VFNOM VFINIT) NIL))
 (PMANSENSED0 ((PMANPREG PMANMAX) NIL))
 (PHESENSED1 ((PHEPREG NIL)) (PULLSENSED1 (0 NIL))
 (VOLFUELSENSED1 ((VFLOW VFNOM) NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
 (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
 (MANVALVE1 (OPEN NIL)))
((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
 (VOLFUELSENSED0 ((VFNOM VFINIT) NIL))
 (PMANSENSED0 ((PMANPREG PMANMAX) NIL))
 (PHESENSED1 ((PHELOW PHEPREG) NIL)) (PULLSENSED1 (0 NIL))
 (VOLFUELSENSED1 ((VFLOW VFNOM) NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
 (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
 (MANVALVE1 (OPEN NIL)))
((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
 (VOLFUELSENSED0 ((VFNOM VFINIT) NIL))
 (PMANSENSED0 ((PMANPREG PMANMAX) NIL))
 (PHESENSED1 ((PHELOW PHEPREG) NIL)) (PULLSENSED1 (0 NIL))
 (VOLFUELSENSED1 (VFLOW NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
 (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
 (MANVALVE1 (OPEN NIL)))
((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
 (VOLFUELSENSED0 ((VFNOM VFINIT) NIL))
 (PMANSENSED0 ((PMANPREG PMANMAX) NIL))
```

```
 (PHESENSED1 ((PHELOW PHEPREG) NIL)) (PULLSENSED1 (0 NIL))
 (VOLFUELSENSED1 ((0 VFLOW) NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
 (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
 (MANVALVE1 (OPEN NIL)))
((PHESENSED0 (PHEINIT NIL)) (PULLSENSED0 ((PULLPREG PULLSREG) NIL))
 (VOLFUELSENSED0 ((VFNOM VFINIT) NIL))
 (PMANSENSED0 ((PMANPREG PMANMAX) NIL))
 (PHESENSED1 ((PHELOW PHEPREG) NIL)) (PULLSENSED1 (0 NIL))
 (VOLFUELSENSED1 (0 NIL)) (PMANSENSED1 ((0 PMANPREG) NIL))
 (TANKVALVE0 (OPEN NIL)) (MANVALVE0 (OPEN NIL)) (TANKVALVE1 (OPEN NIL))
 (MANVALVE1 (OPEN NIL))))
```

Here is the actual QDOCS trace resulting from running QDOCS on the above sequence of sensor values and the model from Appendix A. At each loop iteration, the following information is listed:

- The iteration number.
- The current agenda item (hypothesis) to be considered.
- Whether Check-Hypothesis was called, or if an existing conflict was found that is unhit by the hypothesis.
- The conflict returned, or to be used.
- The agenda of hypotheses to be considered next. For brevity, this is reduced to the first ten hypotheses on the agenda.
- The final hypotheses that have been discovered to be consistent with the observations.

```
Loop number 1
Current Item is: #S(AGENDA-ITEM HYPOTHESIS NIL
                                PROBABILITY 0.7278053637711427)
Calling Check-Hypothesis
The conflict returned is: ((HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                           (SREG-MODE0 (NORMAL STD))
                           (PREG-MODE0 (NORMAL STD))
                           (ULLAGE-PRESSURE-SENSOR-MODE0 (NORMAL STD)))
```

```
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 0.014853170689206998)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 0.014853170689206995)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497))
Final hypotheses now:
NIL
Loop number 2
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE0
                                             . STUCK-AT-0))
                                PROBABILITY 0.014853170689206998)
Calling Check-Hypothesis
The conflict returned is: ((SREG-MODE0 (NORMAL STD))
                           (HELIUM-SYSTEM-MODE0 (NORMAL STD))
                           (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                           (PREG-MODE0 (NORMAL STD))
                           (FUEL-VOLUME-SENSOR-MODE0 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 0.014853170689206995)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((HELIUM-SYSTEM-MODE0 . LEAKING)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.031259324327958E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0 . STUCK-AT-0)
```

```
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 ...)
Final hypotheses now:
NIL
Loop number 3
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0
                                             . STUCK-AT-0))
                                PROBABILITY 0.014853170689206995)
Calling Check-Hypothesis
The conflict returned is: ((SREG-MODE0 (NORMAL STD))
                           (PREG-MODE0 (NORMAL STD))
                           (ULLAGE-PRESSURE-SENSOR-MODE0 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((HELIUM-SYSTEM-MODE0 . LEAKING)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.031259324327958E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
```

```
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 ...)
Final hypotheses now:
NIL
Loop number 4
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN))
                                PROBABILITY 0.007426585344603497)
Calling Check-Hypothesis
The conflict returned is: ((HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                           (SREG-MODE0 (STUCK-OPEN STD))
                           (PREG-MODE0 (NORMAL STD))
                           (ULLAGE-PRESSURE-SENSOR-MODE0 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((HELIUM-SYSTEM-MODE0 . LEAKING)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.031259324327958E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 ...)
```

```
Final hypotheses now:
NIL
Loop number 5
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED))
                                PROBABILITY 0.007426585344603497)
Calling Check-Hypothesis
The conflict returned is: ((PREG-MODE0 (NORMAL STD))
                          (ULLAGE-PRESSURE-SENSOR-MODE0 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((HELIUM-SYSTEM-MODE0 . LEAKING)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.031259324327958E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 ...)
Final hypotheses now:
NIL
Loop number 6
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN))
                                PROBABILITY 0.007426585344603497)
```

```
Calling Check-Hypothesis
The conflict returned is: ((HE-TANK-P-SENSOR-MODE1 (NORMAL STD))
                          (SREG-MODE1 (NORMAL STD))
                          (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                          (SREG-MODE0 (NORMAL STD))
                          (PREG-MODE1 (NORMAL STD))
                          (ULLAGE-PRESSURE-SENSOR-MODE1 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 0.007426585344603497)
 #S(AGENDA-ITEM HYPOTHESIS ((HELIUM-SYSTEM-MODE0 . LEAKING)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.031259324327958E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 ...)
Final hypotheses now:
NIL
Loop number 7
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED))
                                PROBABILITY 0.007426585344603497)
```

```
Using conflict: ((HE-TANK-P-SENSOR-MODE1 (NORMAL STD))
                 (SREG-MODE1 (NORMAL STD))
                 (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                 (SREG-MODE0 (NORMAL STD))
                 (PREG-MODE1 (NORMAL STD))
                 (ULLAGE-PRESSURE-SENSOR-MODE1 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((HELIUM-SYSTEM-MODE0 . LEAKING)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.031259324327958E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 ...)
Final hypotheses now:
NIL
Loop number 8
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((HELIUM-SYSTEM-MODE0 . LEAKING)
                                            (ULLAGE-PRESSURE-SENSOR-MODE0
```

```
                                          . STUCK-AT-0))
                                PROBABILITY 3.031259324327958E-4)
Using conflict: ((HE-TANK-P-SENSOR-MODE1 (NORMAL STD))
                 (SREG-MODE1 (NORMAL STD))
                 (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                 (SREG-MODE0 (NORMAL STD))
                 (PREG-MODE1 (NORMAL STD))
                 (ULLAGE-PRESSURE-SENSOR-MODE1 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 ...)
Final hypotheses now:
NIL
Loop number 9
```

```
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE0
                                             . STUCK-AT-0)
                                            (ULLAGE-PRESSURE-SENSOR-MODE0
                                             . STUCK-AT-0))
                                PROBABILITY 3.0312593243279575E-4)
Calling Check-Hypothesis
The conflict returned is: ((SREG-MODE0 (NORMAL STD))
                           (PREG-MODE0 (NORMAL STD))
                           (HE-TANK-P-SENSOR-MODE0 (STUCK-AT-0 STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0 . STUCK-AT-0)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 3.0312593243279575E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 ...)
Final hypotheses now:
NIL
```

```
Loop number 10
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((FUEL-VOLUME-SENSOR-MODE0
                                             . STUCK-AT-0)
                                            (ULLAGE-PRESSURE-SENSOR-MODE0
                                             . STUCK-AT-0))
                                PROBABILITY 3.0312593243279575E-4)
Using conflict: ((HE-TANK-P-SENSOR-MODE1 (NORMAL STD))
                 (SREG-MODE1 (NORMAL STD))
                 (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                 (SREG-MODE0 (NORMAL STD))
                 (PREG-MODE1 (NORMAL STD))
                 (ULLAGE-PRESSURE-SENSOR-MODE1 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
```

```
 ...)
Final hypotheses now:
NIL
Loop number 11
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                                            (ULLAGE-PRESSURE-SENSOR-MODE0
                                             . STUCK-AT-0))
                                PROBABILITY 1.5156296621639793E-4)
Using conflict: ((HE-TANK-P-SENSOR-MODE1 (NORMAL STD))
                 (SREG-MODE1 (NORMAL STD))
                 (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                 (SREG-MODE0 (NORMAL STD))
                 (PREG-MODE1 (NORMAL STD))
                 (ULLAGE-PRESSURE-SENSOR-MODE1 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639793E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
```

```
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639788E-4)
 ...)
Final hypotheses now:
NIL
Loop number 12
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                                            (ULLAGE-PRESSURE-SENSOR-MODE0
                                             . STUCK-AT-0))
                                 PROBABILITY 1.5156296621639793E-4)
Using conflict: ((HE-TANK-P-SENSOR-MODE1 (NORMAL STD))
                 (SREG-MODE1 (NORMAL STD))
                 (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                 (SREG-MODE0 (NORMAL STD))
                 (PREG-MODE1 (NORMAL STD))
                 (ULLAGE-PRESSURE-SENSOR-MODE1 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
```

```
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.5156296621639788E-4)
 ...)
Final hypotheses now:
NIL
Loop number 13
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-OPEN)
                                            (HE-TANK-P-SENSOR-MODE0
                                             . STUCK-AT-0))
                                PROBABILITY 1.515629662163979E-4)
Calling Check-Hypothesis
The conflict returned is: ((HE-TANK-P-SENSOR-MODE0 (STUCK-AT-0 STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-OPEN)
```

```
                          (PREG-MODE0 . STUCK-CLOSED))
               PROBABILITY 7.578148310819897E-5)
 ...)
Final hypotheses now:
NIL
Loop number 14
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE0 . STUCK-CLOSED)
                                            (HE-TANK-P-SENSOR-MODE0
                                             . STUCK-AT-0))
                                 PROBABILITY 1.515629662163979E-4)
Using conflict: ((HE-TANK-P-SENSOR-MODE0 (STUCK-AT-0 STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                          (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                          (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                          (PREG-MODE0 . STUCK-CLOSED))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                          (PREG-MODE0 . STUCK-CLOSED))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                          (PREG-MODE0 . STUCK-OPEN))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                          (PREG-MODE0 . STUCK-OPEN))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                          (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                          (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
               PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-OPEN)
                          (PREG-MODE0 . STUCK-CLOSED))
               PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-CLOSED)
                          (PREG-MODE0 . STUCK-CLOSED))
               PROBABILITY 7.578148310819897E-5)
 ...)
```

```
Final hypotheses now:
NIL
Loop number 15
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                                            (ULLAGE-PRESSURE-SENSOR-MODE0
                                             . STUCK-AT-0))
                                PROBABILITY 1.515629662163979E-4)
Calling Check-Hypothesis
The conflict returned is: ((ULLAGE-PRESSURE-SENSOR-MODE0 (STUCK-AT-0 STD))
                           (HELIUM-SYSTEM-MODE0 (NORMAL STD))
                           (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                           (PREG-MODE0 (NORMAL STD))
                           (FUEL-VOLUME-SENSOR-MODE0 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (ULLAGE-PRESSURE-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.515629662163979E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-OPEN)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-CLOSED)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-OPEN)
                            (PREG-MODE0 . STUCK-OPEN))
```

```
               PROBABILITY 7.578148310819897E-5)
 ...)
Final hypotheses now:
NIL
Loop number 16
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                                            (ULLAGE-PRESSURE-SENSOR-MODE0
                                             . STUCK-AT-0))
                                PROBABILITY 1.515629662163979E-4)
Using conflict: ((ULLAGE-PRESSURE-SENSOR-MODE0 (STUCK-AT-0 STD))
                 (HELIUM-SYSTEM-MODE0 (NORMAL STD))
                 (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                 (PREG-MODE0 (NORMAL STD))
                 (FUEL-VOLUME-SENSOR-MODE0 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-OPEN)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-CLOSED)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-OPEN)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-CLOSED)
```

```
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 7.578148310819897E-5)
 ...)
Final hypotheses now:
NIL
Loop number 17
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1
                                             . STUCK-AT-0)
                                            (PREG-MODE0 . STUCK-CLOSED))
                                 PROBABILITY 1.5156296621639788E-4)
Calling Check-Hypothesis
The conflict returned is: ((HELIUM-SYSTEM-MODE1 (NORMAL STD))
                           (FUEL-VOLUME-SENSOR-MODE1 (NORMAL STD))
                           (SREG-MODE1 (NORMAL STD))
                           (PREG-MODE0 (STUCK-CLOSED STD))
                           (HELIUM-SYSTEM-MODE0 (NORMAL STD))
                           (HE-TANK-P-SENSOR-MODE0 (NORMAL STD))
                           (SREG-MODE0 (NORMAL STD))
                           (PREG-MODE1 (NORMAL STD))
                           (ULLAGE-PRESSURE-SENSOR-MODE1 (NORMAL STD)))
New Agenda:
(#S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((HE-TANK-P-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-OPEN)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE0 . STUCK-CLOSED)
                            (HE-TANK-P-SENSOR-MODE0 . STUCK-AT-0))
                PROBABILITY 1.5156296621639788E-4)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-OPEN)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-CLOSED)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-OPEN)
```

```
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((PREG-MODE1 . STUCK-CLOSED)
                            (PREG-MODE0 . STUCK-OPEN))
                PROBABILITY 7.578148310819897E-5)
 #S(AGENDA-ITEM HYPOTHESIS ((SREG-MODE1 . STUCK-OPEN)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 7.578148310819895E-5)
 ...)
Final hypotheses now:
NIL
Loop number 18
Current Item is: #S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1
                                             . STUCK-AT-0)
                                            (PREG-MODE0 . STUCK-CLOSED))
                                PROBABILITY 1.5156296621639788E-4)
Calling Check-Hypothesis
The conflict returned is: NIL
Final hypotheses now:
(#S(AGENDA-ITEM HYPOTHESIS ((ULLAGE-PRESSURE-SENSOR-MODE1 . STUCK-AT-0)
                            (PREG-MODE0 . STUCK-CLOSED))
                PROBABILITY 1.5156296621639788E-4))
```

Appendix C

The Level-Controlled Tank

This is the QSIM model followed by the `defComponents` and `defOperatingRegions` descriptions of the Level-Controlled Tank model.

```
(define-qde controlled-tank-model
  (text "tank with controller")
  (quantity-spaces
   (height    (0 low norm max))         ; height of water
   (height-sensed (0 low norm max))     ; sensed height of water
   (float-contact-voltage (0 low norm max))
                                        ; voltage at contact with float
   (float-contact (0 low norm max))     ; point of contact of float
   (setting (0 low norm max))           ; desired level setting
   (setting-contact (0 low norm max))   ; point of contact at setting
   (setting-contact-voltage (0 low norm max))
                                        ; voltage at setting contact
   (battery-voltage (0 high))           ; battery voltage
   (hvoltage (0 low norm max))          ; voltage due to float level
   (rvoltage (0 low norm max))          ; voltage due to setting
   (error (mhigh 0 high))               ; difference in voltage between
                                        ; float and setting.
   (measured-volts (mhigh 0 high))      ; voltmeter's measurement
   (ampl-gain (0 Ka max))               ; Gain factor of amplifier
   (Voltout (mhigh 0 high))             ; Output of amplifier
   (Motorvolts (mhigh 0 high))          ; Voltage at motor
   (Omegam (mhigh 0 high))              ; Rotational speed of motor
   (Thetam (0 thetammax))               ; Rotational position of motor
   (current (mhigh 0 high))             ; Current through the motor
   (current-sensed (mhigh 0 high))      ; Current sensed at ammeter
   (inflowrate (0 flowmax))             ; Inflow into the tank
```

```
 (outflowrate (0 low norm high outflowmax))
                                      ; Outflow from the tank
 (netflow (mhigh 0 high)))            ; Net flow rate from the tank

(discrete-variables
 (float-sensor-mode (normal stuck-low stuck-high))
                                      ; Mode of float sensor
 (float-contact-mode (normal stuck-low stuck-high))
                                      ; Mode of float contact
 (battery (normal dead))
                                      ; Mode of battery
 (setting-amplifier-connection (normal broken))
                                      ; Mode of setting->amplifier
                                      ; connection
 (setting-contact-mode (normal stuck-low stuck-high))
                                      ; Mode of setting contact
 (float-amplifier-connection (normal broken))
                                      ; Mode of float-contact->amp
                                      ; connection
 (amplifier-mode (normal no-connection))
                                      ; Mode of amplifier
 (amplifier-motor-connection (normal broken))
                                      ; Mode of amp-motor conn.
 (ammeter-mode (normal stuck-at-0 broken))
                                      ; Mode of ammeter
 (voltmeter (normal stuck-at-0))      ; Mode of voltmeter
 (motor-mode (normal stuck))          ; Mode of motor
 (valve-mode (normal stuck-open stuck-closed))
                                      ; Mode of reservoir outlet valve
 (water-outflow-mode (normal stuck-closed))
                                      ; Mode of outflow from tank
 (level-sensor-mode (normal stuck-at-0 stuck-at-full))
                                      ; Mode of level sensor

 ;; operating regions for motor

 (motor-region (zero moving saturated)))

(constraints
 ;; Float and contact constraints
 (mode (float-contact-mode normal)
       ((m+ height float-contact)
        (0 0) (low low) (norm norm) (max max)))
```

```
(mode (float-contact-mode stuck-low)
      ((zero-std float-contact )))
(mode (float-contact-mode stuck-high)
      ((constant float-contact max)))
(mode (float-sensor-mode normal)
      ((m+ height height-sensed)
       (0 0) (low low) (norm norm) (max max)))
(mode (float-sensor-mode stuck-low)
      ((zero-std height-sensed)))
(mode (float-sensor-mode stuck-high)
      ((constant height-sensed max)))

;; Electrical constraints at battery and contacts
(mode (battery normal)
      ((constant battery-voltage high)))
(mode (battery dead)
      ((zero-std battery-voltage)))
((mult battery-voltage float-contact float-contact-voltage)
 (0 0 0) (high 0 0) (high low low) (high norm norm) (high max max))
(mode (float-amplifier-connection normal)
      ((equal float-contact-voltage hvoltage)
       (0 0) (low low) (norm norm) (max max)))
(mode (float-amplifier-connection broken)
      ((zero-std hvoltage)))
((constant setting)) ;; shouldn't this be in some fault mode?
(mode (setting-contact-mode normal)
      ((equal setting setting-contact)
       (0 0) (low low) (norm norm) (max max)))
(mode (setting-contact-mode stuck-low)
      ((zero-std setting-contact)))
(mode (setting-contact-mode stuck-high)
      ((constant setting-contact max)))
((mult battery-voltage setting-contact setting-contact-voltage)
 (0 0 0) (high 0 0) (high low low) (high norm norm) (high max max))

;; Contacts to Amplifier and Voltmeter related constraints
(mode (setting-amplifier-connection normal)
      ((equal setting-contact-voltage rvoltage)
       (0 0) (low low) (norm norm) (max max)))
(mode (setting-amplifier-connection broken)
      ((zero-std rvoltage)))
((add error hvoltage rvoltage)
 (mhigh max 0) (0 0 0) (0 max max) (0 low low) (0 norm norm)
```

```
 (high 0 max))
(mode (voltmeter normal)
      ((equal error measured-volts)
       (mhigh mhigh) (0 0) (high high)))
(mode (voltmeter stuck-at-0)
      ((zero-std measured-volts)))
(mode (amplifier-mode normal)
      ((constant ampl-gain Ka)))
(mode (amplifier-mode no-connection)
      ((zero-std ampl-gain)))

;; Output of amplifier and operation of motor
((mult ampl-gain error Voltout)
 (Ka 0 0) (Ka mhigh mhigh) (Ka high high))
(mode (and (amplifier-motor-connection normal)
           (or (ammeter-mode normal) (ammeter-mode stuck-at-0)))
      ((equal Voltout Motorvolts)
       (mhigh mhigh) (0 0) (high high)))
(mode (or (amplifier-motor-connection broken)
          (ammeter-mode broken))
      ((zero-std motorvolts)))
(mode (and (motor-mode normal) (motor-region zero))
      ((less-than-or-equal Motorvolts 0)))
(mode (and (motor-mode normal) (motor-region saturated))
      ((greater-than-or-equal Motorvolts 0)))
(mode (or (motor-mode stuck)
          (and (motor-mode normal)
               (or (motor-region zero)
                   (motor-region saturated))))
      ((zero-std Omegam)))
(mode (and (motor-mode normal) (motor-region moving))
      ((m+ motorvolts omegam)
       (mhigh mhigh) (0 0) (high high)))
(mode (and (motor-mode normal) (motor-region zero))
      ((zero-std Thetam)))
(mode (and (motor-mode normal) (motor-region saturated))
      ((constant Thetam thetammax)))
((d/dt Thetam Omegam))

;; Operation of ammeter
(mode (and (amplifier-motor-connection normal)
           (or (ammeter-mode normal)
               (ammeter-mode stuck-at-0)))
```

```
      ((m+ motorvolts current)
       (mhigh mhigh) (0 0) (high high)))
 (mode (or (amplifier-motor-connection broken)
           (ammeter-mode broken))
      ((zero-std current)))
 (mode (ammeter-mode normal)
      ((equal current current-sensed)
       (mhigh mhigh) (0 0) (high high)))
 (mode (or (ammeter-mode stuck-at-0)
           (ammeter-mode broken))
      ((zero-std current-sensed)))

 ;; Operation of outlet valves from reservoir
 (mode (valve-mode normal)
      ((m+ thetam inflowrate)
       (0 0) (thetammax flowmax)))
 (mode (valve-mode stuck-open)
      ((constant inflowrate flowmax)))
 (mode (valve-mode stuck-closed)
      ((zero-std inflowrate)))

 ;; Tank inflow and outflow
 (mode (water-outflow-mode normal)
      ((m+ height outflowrate)
       (max outflowmax)
       (norm norm)
       (low low)
       (0 0)))
 (mode (water-outflow-mode stuck-closed)
      ((zero-std outflowrate)))
 ((add netflow outflowrate inflowrate)
  (0 0 0) (mhigh outflowmax 0) (high 0 flowmax)
  (0 outflowmax flowmax))
 ((d/dt height netflow)))

;; Region transitions of motor.
(transitions
 ((and (motor-mode (normal nil)) (motor-region (zero nil))
      (motorvolts (0 inc)))
  -> start-open)
 ((and (motor-mode (normal nil)) (motor-region (moving nil))
      (thetam (thetammax nil))
      (or (motorvolts ((0 high) nil))
```

```
            (motorvolts (high nil))))
    -> saturate)
   ((and (motor-mode (normal nil)) (motor-region (saturated nil))
         (motorvolts (0 dec)))
    -> start-close)
   ((and (motor-mode (normal nil)) (motor-region (moving nil))
         (thetam (0 nil))
         (or (motorvolts (mhigh nil))
             (motorvolts ((mhigh 0) nil))))
    -> stop-motor)))



;;; Region Transition functions

(defun start-open (state)
  (create-transition-state
   :from-state state
   :to-qde controlled-tank-model
   :assert '((motor-region (moving std)))
   :inherit-qmag #'all-except-omegam
   :inherit-qdir nil)) ;; should this really be nil??

(defun saturate (state)
  (create-transition-state
   :from-state state
   :to-qde controlled-tank-model
   :assert '((motor-region (saturated std)))
   :inherit-qmag #'all-except-omegam
   :inherit-qdir nil)) ;; should this really be nil??

(defun start-close (state)
  (create-transition-state
   :from-state state
   :to-qde controlled-tank-model
   :assert '((motor-region (moving std)))
   :inherit-qmag #'all-except-omegam
   :inherit-qdir nil)) ;; should this really be nil??


(defun stop-motor (state)
  (create-transition-state
   :from-state state
```

```
   :to-qde controlled-tank-model
   :assert '((motor-region (zero std)))
   :inherit-qmag #'all-except-omegam
   :inherit-qdir nil)) ;; should this really be nil??


(defun all-except-omegam (varname)
  (not (eq varname 'omegam)))


;;; Component structure for Controlled Tank model

(defComponents controlled-tank-model
  (float-sensor float-sensor-mode (normal . 0.98) (stuck-low . 0.01)
                (stuck-high . 0.01))
  (float-contact float-contact-mode (normal . 0.98) (stuck-low . 0.01)
                 (stuck-high . 0.01))
  (setting-contact setting-contact-mode (normal . 0.98)
                   (stuck-low . 0.01) (stuck-high . 0.01))
  (battery battery (normal . 0.98) (dead . 0.02))
  (setting-amplifier-connection setting-amplifier-connection
                                (normal . 0.98) (broken . 0.02))
  (float-amplifier-connection float-amplifier-connection
                              (normal . 0.98) (broken . 0.02))
  (amplifier amplifier-mode (normal . 0.98) (no-connection . 0.02))
  (amplifier-motor-connection amplifier-motor-connection
                              (normal . 0.98) (broken . 0.02))
  (motor motor-mode (normal . 0.98) (stuck . 0.02))
  (water-outflow water-outflow-mode (normal . 0.98) (stuck-closed . 0.02))
  (voltmeter voltmeter (normal . 0.99) (stuck-at-0 . 0.01))
  (ammeter ammeter-mode (normal . 0.99) (stuck-at-0 . 0.005)
           (broken . 0.005))
  (valve valve-mode (normal . 0.98) (stuck-open . 0.01)
         (stuck-closed . 0.01))
  (level-sensor level-sensor-mode (normal . 0.99) (stuck-at-0 . 0.005)
                (stuck-at-full . 0.005)))


(defOperatingRegions controlled-tank-model
  ((motor-mode . normal) (motor-region . zero) (motor-region . saturated)
   (motor-region . moving))
  ((motor-mode . stuck) (motor-region . moving)))
```

Bibliography

Abbott, K. H. [1988]. Robust operative diagnosis as problem solving in a hypothesis space. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 369–374 Minneapolis, MN.

Bousson, K., Zimmer, L., & Travé-Massuyès, L. [1994]. Causal model-based diagnosis of dynamic systems. In *Fifth International Workshop on Principles of Diagnosis*, pp. 34–41 New Paltz, NY.

Catino, C. A. [1993]. *Automated Modeling of Chemical Plants with Application to Hazard and Operability Studies*. Ph.D. thesis, Department of Chemical Engineering, University of Pennsylvania.

Dague, P., Jehl, O., Deves, P., Luciani, P., & Taillibert, P. [1991]. When oscillators stop oscillating. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 1109–1115 Sydney, Australia.

Dalle Molle, D. [1989]. Qualitative simulation of dynamic chemical processes. Tech. rep. AI89–107, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas 78712.

Davis, R. [1984]. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, *24*, 347–410.

de Kleer, J. [1986]. An assumption-based TMS. *Artificial Intelligence*, *28*, 127–162.

de Kleer, J., & Williams, B. C. [1987]. Diagnosing multiple faults. *Artificial Intelligence*, *32*, 97–130.

de Kleer, J., & Williams, B. C. [1989]. Diagnosis with behavioral modes. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 1324–1330 Detroit, MI.

deKleer, J., & Brown, J. [1984]. A qualitative physics based on confluences. *Artificial Intelligence*, *24*, 7–83.

deKleer, J. [1991]. Focusing on probable diagnoses. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 842–848 Anaheim, CA.

Doyle, J. [1979]. A truth maintenance system. *Artificial Intelligence*, *12*, 231–272.

Doyle, R. J., Sellers, S. M., & Atkinson, D. J. [1989]. A focused, context-sensitive approach to monitoring. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 1231–1237 Detroit, MI.

Dressler, O. [1994]. Model-based diagnosis on board: Magellan-MT inside. In *Fifth International Workshop on Principles of Diagnosis*, pp. 87–92 New Paltz, NY.

Dressler, O., & Struss, P. [1992]. Back to defaults: Characterizing and computing diagnoses as coherent assumption sets. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 719–723.

Dressler, O., & Struss, P. [1994]. Model-based diagnosis with the default-based diagnosis engine: Effective control strategies that work in practice. In *Fifth International Workshop on Principles of Diagnosis*, pp. 93–97 New Paltz, NY.

Dvorak, D. [1992]. *Monitoring and Diagnosis of Continuous Dynamic Systems Using Semiquantitative Simulation*. Ph.D. thesis, University of Texas, Austin, TX.

Dvorak, D., & Kuipers, B. [1992]. Model-based monitoring of dynamic systems. In Hamscher, W., Console, L., & de Kleer, J. (Eds.), *Readings in Model-Based Diagnosis*, pp. 249–254. Morgan Kaufmann, San Mateo, CA.

Farquhar, A. [1993]. *Automated Modeling of Physical Systems in the Presence of Incomplete Knowledge*. Ph.D. thesis, Artificial Intelligence Laboratory, University of Texas. Technical Report AI93-207.

Forbus, K. D. [1984]. Qualitative process theory. *Artificial Intelligence*, *24*, 85–168.

Genesereth, M. [1984]. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, *24*, 411–436.

Hamscher, W., Console, L., & deKleer, J. (Eds.). [1992]. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, San Mateo, CA.

Kapadia, R., Biswas, G., & Robertson, C. [1994]. Doc: A framework for monitoring and diagnosis of continuous-valued systems. In *Fifth International Workshop on Principles of Diagnosis*, pp. 140–147 New Paltz, NY.

Kay, H. [1992]. A qualitative model of the space shuttle reaction control system. Tech. rep. AI92-188, Artificial Intelligence Laboratory, University of Texas, Austin, TX.

Kay, H., & Kuipers, B. J. [1993]. Numerical behavior envelopes for qualitative models. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 606–613 Washington, D.C.

Kuipers, B. J., & Berleant, D. [1988]. Using incomplete quantitative knowledge in qualitative reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 324–329 St. Paul, MN.

Kuipers, B. J. [1984]. Commonsense reasoning about causality: Deriving behavior from structure. *Artificial Intelligence*, *24*, 169–203.

Kuipers, B. J. [1994]. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, Cambridge, MA.

Kuo, B. C. [1991]. *Automatic Control Systems*. Prentice Hall, Engelwood Cliffs, New Jersey.

Lackinger, F., & Nejdl, W. [1991]. Integrating model-based monitoring and diagnosis of complex dynamic systems. In *Proceedings of the Twelfth International Joint Conference on Artificial intelligence*, pp. 1123–1128 Sydney, Australia.

Minton, S. N. [1988]. *Learning Effective Search Control Knowledge: An Explanantion-Based Approach*. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, PA.

Ng, H. T. [1990]. Model-based, multiple fault diagnosis of time-varying, continuous physical devices. In *Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications*, pp. 9–15 Santa Barbara, CA. Reprinted in *Readings in Model-based Diagnosis*, W. Hamscher, L. Console, and J. de Kleer (eds.), Morgan Kaufman, San Mateo, CA, 1992.

Ng, H. T. [1991]. Model-based, multiple-fault diagnosis of dynamic, continuous physical devices. *IEEE Expert*, *6*[6], 38–43.

Ng, H. T. [1992]. *A General Abductive System with Applications to Plan Recognition and Diagnosis*. Ph.D. thesis, University of Texas, Austin, TX. Also appears as Artificial Intelligence Laboratory Technical Report AI 92-177.

Nilsson, N. [1980]. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

Oyeleye, O. O., Finch, F. E., & Kramer, M. A. [1990]. Qualitative modeling and fault diagnosis of dynamic processes by midas. *Chemical Engineering Communications*, *96*, 205–228.

Poole, D. [1989]. Normality and faults in logic-based diagnosis. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 1304–1310 Detroit, MI.

Reggia, J. A., Nau, D. S., & Wang, P. Y. [1983]. Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, *19*, 437–460.

Reiter, R. [1987]. A theory of diagnosis from first principles. *Artificial Intelligence*, *32*, 57–95.

Reiter, R., & de Kleer, J. [1987]. Foundations of assumption-based truth maintenance systems. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 183–188.

Rickel, J. [1995]. *Automated Modeling of Complex Systems to Answer Prediction Questions*. Ph.D. thesis, Department of Computer Science, University of Texas at Austin.

Shortliffe, E., & Buchanan, B. [1975]. A model of inexact reasoning in medicine. *Mathematical Biosciences*, *23*, 351–379.

Struss, P., & Dressler, O. [1989]. Physical negation — integrating fault models into the general diagnostic engine. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 1318–1323 Detroit, MI.

Subramanian, S., & Mooney, R. J. [1994]. Multiple-fault diagnosis using general qualitative models with fault modes. In *Fifth International Workshop on Principles of Diagnosis*, pp. 321–325 New Paltz, NY.

Subramanian, S., & Mooney, R. J. [1995]. Multiple-fault diagnosis using qualitative models and fault modes. In *IJCAI-95 Workshop on Engineering Problems in Qualitative Reasoning* Montreal, Quebec, Canada.

Vinson, J. M., & Ungar, L. H. [1995]. Dynamic process monitoring and fault diagnosis with qualitative models. *IEEE Transactions on Systems, Man, and Cybernetics*, *25*[1].

Waltz, D. [1975]. Understanding line drawings of scenes with shadows. In Winston, P. H. (Ed.), *The Psychology of Computer Vision*, pp. 19–91. McGraw Hill, Cambridge, Mass.

Vita

Siddarth Subramanian was born in Beijing, China in 1965, the son of Gita Subramanian and Chalakudi Narayaniyer Subramanian. In 1983, he graduated from the International School of Islamabad in Islamabad, Pakistan. He then pursued a degree in Computer Sciences at the Indian Institute of Technology, in Kanpur, India. He graduated with a Bachelor of Technology in 1987 and was awarded a Microelectronics and Computer Development Fellowship at the University of Texas at Austin for graduate work in Computer Sciences. In 1989, he was awarded a Master of Science in Computer Science from the University.

Permanent Address: 1507 Ruth Ave.
Austin, TX 78757
United States of America

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin.