SURE 静岡大学学術リポジトリ
Shizuoka University REpository

A Framework for Model-Based Diagnostic Expert Systems

メタデータ	言語: en 出版者: Shizuoka University 公開日: 2012-03-08 キーワード (Ja): キーワード (En): 作成者: Yusuf, Wilajati Purna メールアドレス: 所属:
URL	https://doi.org/10.11501/3111327

電子科学研究科Y GD K
0002513554 R 137
静岡大学附属図書館

A Framework for Model-Based Diagnostic Expert Systems

(モデルに基づく診断型エキスパートシステムの枠組みに関する研究)

by

Yusuf Wilajati Purna

静岡大学 寄贈 附属図書館
静岡大学図書

DISSERTATION

Presented to the Faculty of the
Graduate School of Electronic Science & Technology
Shizuoka University
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Graduate School of Electronic Science and Technology
Shizuoka University

January, 1996

To
my parents,
brothers & sisters,
Kieko Kumagai, and
Rina Akitia

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor, Professor Takahira Yamaguchi, for his guidance, continuing encouragement, and valuable discussions throughout this research. I owe much Professor Tadashii Yanagizawa for the rather detailed explanation on the basic mechanism of refrigeration plants and for improving and correcting the examples of possible faults in the domain. I would also like to thank Professor Tadanori Mizuno and Professor Atsuyuki Suzuki for their helpful suggestions and comments on the research. In addition, to all professors at the Department of Computer Science, Faculty of Engineering, Graduate School of Engineering, and Graduate School of Electronic Science & Technology, Shizuoka University, who taught me most of what I know about computer science, I am very indebted.

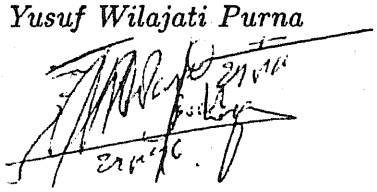
My thanks also go to all members of Yamaguchi's laboratory. Their help, discussions, camaraderie, and fellowship are useful support in carrying out this research. Yoshiaki Tachibana always provides helpful information on the computer system, which I lack much. Thanks a lot. Masaki Kurematu keeps me informed important information in the university which I usually ignore. Thank you. I thank Torikoshi, an ex-member of Yamaguchi's research group, for allowing me to use his fault tree generation program.

I sincerely appreciate the financial assistance of the OFP and STAID program at the Agency for the Assessment & Application of Technology Indonesia (BPP Teknologi), without which I would never have started my study in Japan. The

almost-ten-year-continuous scholarship has really supported my live and study in Japan.

I would also like to thank my parents, brothers, and sisters, without whom I would never have started this research. Finally, but importantly, I would like to thank my wife, Kieko Kumagai, my daughter, Rina Akitia, for their patience, love, and encouraging me when I thought that the task was too great. Their support has been paramount for carrying through the research.

Yusuf Wilajati Purna



Graduate School of Electronic Science and Technology

Shizuoka University

January 1996

Preface

Having studied artificial intelligence (AI) and expert systems (ES) for a few years, I become to know how difficult to systematize humans' way of thinking in solving a problem into an electronic computer. In order to be able to do the task, at least we are demanded to provide explicit descriptions of *knowledge* and *reasoning methods* required to carry out the task. For example, to enable a computer to do a simple task such as the task of changing the order of stacked blocks from a certain state to another one (known as *A Simple Blocks World Problem* in AI literature), we need to provide those what the AI community calls *state descriptions, operators, predicates,* and *rules for reasoning*, which actually we as human beings can be considered not to use in doing the task. Furthermore, to make a computer able to do more complicated tasks such as diagnosis, planning, scheduling, etc., in other words, to build an expert system, we have to take into account *sorts* of knowledge and reasoning methods besides the aforementioned. Most of the heart of AI research on automated reasoning is centered on investigating those appropriate descriptions of knowledge and reasoning methods, and on identifying useful kinds of them. My work in the doctor course is not an exception.

In this thesis, I describe my ongoing research on providing a framework for diagnosis which is one of the major application areas of AI and ES. To carry out the task of diagnosis, in principle I lay emphasis on recognizing what kinds of subtasks the task can be composed of, and what sorts of knowledge and strategies can be

employed in each subtask. Extending one of the most well-known general-purpose control strategies in AI, *Generate-and-Test Paradigm*, I divide the task of diagnosis into three subtasks: *Hypothesis Generation, Hypothesis Testing, and Hypothesis Classification.* Until this time, the end of the doctor course, I have finished almost the three subtasks although some implementations and more experiments still should be done in order to really evaluate the proposed framework. This thesis elaborates on the three subtasks, explaining in detail some useful sorts of knowledge and strategies appropriate to the framework, and algorithms for coordinating them. Besides, several examples of diagnosing faults using the proposed framework are also presented in order to show the diagnosing capability of the framework.

The framework itself is still far from being a real diagnostic expert system. There are lots of outstanding issues that need to be further done. Nevertheless, the essence of the framework promises us a forward step towards providing a robust framework for real diagnostic expert systems.

Abstract

Diagnosis is a significant application of *artificial intelligence* or *expert systems* technology in particular. It is also a hard task since it is required a great deal of various information to do the task. In recent years, research for providing a framework for it has been actively being done. In the field, model-based approach, which exploits the notions of structure and behavior of a device in performing the task, has received a lot of interests of both researchers and practitioners. Although the approach gives a solution to the problem of diagnosing unanticipated faults—faults that need knowledge lying outside the preset knowledge to be able to diagnosed with, it still possesses several drawbacks such as the difficulty in diagnosing structural faults, complex devices, and dynamical faults, and limitationa on symptom representations, due to lack of information outside the model and clear division of the diagnostic task. Therefore, the ultimate goal of the research is to overcome the aforementioned drawbacks, providing a robust framework for diagnostic expert systems. By the way, based closely on the humans' way of accomplishing a diagnostic task, diagnosis as a whole can be divided into three subtasks: *Hypothesis Generation, Hypothesis Testing, and Hypothesis Classification.* Thus in order to fulfill the goal, the whole framework should cover the three subtasks.

This thesis explains frameworks for the three subtasks: hypothesis generation, hypothesis testing and hypothesis classification. The framework classifies information required to diagnose faults into three classes: the *Domain Model, Additional*

Domain Knowledge, and *Diagnostic Strategy.* Three sorts of knowledge belonging to the domain model, namely, *Device Model, Process Model,* and *Topological Relative Position* are used to model the device being diagnosed. *Heuristics*, *Naive Physics*, *Component Durability*, *Component Failure Rates*, and *Component Observability* which are classified into the additional domain knowledge, are employed as additional information to diagnose faults. As strategies for diagnosing faults, the framework applies six sorts of diagnostic strategies: the *Qualitative Value Propagation, Direct Path of Causality, Structural Fault Localization*, *Pseudo Fault Hypothesis Elimination*, *Contradicting Fault Hypothesis Elimination*, and *Candidate Fault Verification.* Algorithms for coordinating the sorts of knowledge and the strategies in diagnosing faults is also provided in the framework.

The framework was implemented in a computer program called MODEST and tested on the domain of refrigeration plants and evaluated using some possible faults in the domain. It showed that it could detect a structural fault that even the current model-based approaches can hardly diagnose. Although there is still a lot of issues remaining to be done, on the whole, the framework is an important breakthrough in model-based diagnosis and should become a step towards providing a framework for diagnostic systems.

Chapter 1

Introduction

1.1 Background and Motivation

As a relatively new discipline science, AI has progressed rapidly in recent years. Not only researchers but also engineers and practitioners in various fields have showed interest in the discipline since it has promised reasonable prospect of solving complicated problems that often they can not deal with. Particularly since research on *knowledge-based systems* as a continuation of AI began actively being done around the seventies, the situation has grown more and more.

Unlike most of research on AI in the sixties which placed emphasis on developing general purpose problem solving techniques , for example GPS[32], which actually could only cope with the so-called *toy* problems[1], knowledge-based systems research attempted to develop methods and techniques, and to incorporate large amounts of relevant task specific knowledge within a domain, in order to solve complex *real* world problems. Expert systems can be considered to be a fruit of the research. In principle, they are built by using an approach taking knowledge from human experts and representing it as a knowledge base, which can then be processed to solve complicated problems in the same way the expert would. Applications of them

[1]The problem mentioned in the preface, a simple blocks world problem, is an example of these problems.

can be found in various domains, performing many different functions, including, diagnosis, planning, monitoring, and scheduling.

Relating to diagnosis, the MYCIN project[84] can be regarded as one of the first successful applications of expert systems. Briefly speaking, MYCIN is a diagnostic expert system for providing therapy advice about certain kinds of infectious diseases. The main feature of this expert system is that MYCIN applies primitive rules in the following form:

if *premise* **then** *consequence*,

to storing knowledge of an expert. Owing to its applying the primitive rules MYCIN can then be also categorized into a rule-based expert system. In that time, such a kind of knowledge representation was popular, and a lot expert systems utilized it due to its simplicity. Even now, the inference engine and the knowledge representation scheme developed for MYCIN is still the basis of many of the current commercial expert system shells.

However, in the early eighties many researchers realized some limitations of such kinds of expert systems (e.g., [22, 45, 56]). The knowledge in these rule-based expert systems was soon referred to as *shallow knowledge* since it was typically derived from experience and case studies of the expert rather than a *deep* understanding of the domain. It only directly states the causality between the "premise" and the "consequence" without referring to the underlying principles. This primarily caused difficulty to the expert diagnostic system in facing a situation at which knowledge lying outside the knowledge preset in the knowledge base is required to do the task, i.e., in coping with any unanticipated symptoms or faults. Intending to cope with such difficulty, a lot of researchers then began turning their attention to the so-called *deep knowledge* and doing research on it. This led to the investigation of *model-*

based approaches[23, 27, 29, 42, 44] and other kinds of approaches such as *functional representation*[14, 80] and *causal networks* [97], which in principle lay emphasis on the underlying principles of the domain rather than the heuristic relation between the premise and the consequence.

Among the deep knowledge based approaches, model-based approaches have been the primest subject to research since they deeply focus on *first principles* of the domain, that is to say, the structure and behavior of the domain. With the principles, they have also proved that they can overcome the difficulty in coping with unanticipated faults. Despite the success, however, they have introduced other problems. It is due to their using first principles that they turn out to fail to diagnose faults that can only be detected by using heuristics and humans' common sense.

Now, the challenging research is then to extend the model-based approach, taking into account such heuristics and common sense. It is this that is the heart of the current research on model-based diagnosis.

1.2 What is Diagnosis ?

In the preceding section, we have touched on shallow and deep knowledge based diagnostic expert systems. But, what is diagnosis ? If we look up the word *diagnose* in *Random House Webster's College Dictionary*[74] we will find that *to diagnose* is *to ascertain the cause or nature (a disorder or problem) from the symptoms*, as one of the general meanings. Based on this, we can mean more clearly that diagnosis is the identification of the cause or causes (usually considered to be *faults*) of some observed undesirable behavior, by means of reasoning based on those observations (generally termed *symptoms*). For example, the identification of disease by means of the patient's symptoms, or the ascertainment of the cause of a mechanical failure.

Diagnosis is essentially an abductive reasoning process, and abduction itself is

the inference of a value, or solution, from known results. Abduction is also often considered as a process of arriving at an explanation[107]. For example, if the following are given:

if *a bulb is blown* **then** *the bulb does not light*,

the **fact** is that *the bulb does not light*,

then using abduction we can infer that the bulb is blown, as an explanation for the fact. This is, however, only one reason for not lighting; it is also possible that there is no electricity flowing to the bulb, which is sometimes a more normal diagnosis. For this reason abduction is also known as plausible inference, but not legal inference.

On the other hand, deduction is legal inference, that is to say, deduction from true premisses is guaranteed to result in a true conclusion. For example, if we know the following:

if *a bulb is blown* **then** *the bulb does not light*,

the **fact** is that *the bulb is blown*,

then using the rule of modus ponens we can conclude that the bulb does not light, based on the fact. However, for diagnostic problems, the fact that is known is the consequence (the symptom), not the premise (the cause), so it is clear that deductive inference, alone, is not fit for such applications.

However, using abduction it is possible to get more than one result, as in the bulb case above. What is needed is information enabling the most likely answer to be chosen. If the probability of alternative answers is known, then the best guess is the most probable. For example, if the probability of being blown is greater than that of electricity's not flowing to the bulb, it can be inferred that the bulb most probably is blown. Therefore, the generation of a diagnosis should be viewed as the careful use of abductive reasoning, considering such kind of information.

1.3 Model-Based Diagnosis

If in identifying possible faults from the observed symptoms, we mainly exploit the information on the *structure* and *behavior* of the domain, then the process is called *model-based diagnosis.* What this kind of reasoning process wants to accentuate is that for the purpose of diagnosing a system, when the system malfunctions, knowing deeply about the system itself is really more important than having the empirical knowledge of diagnosing it. The reason is as follows. Given a symptom, which is usually a deviation from the normal behavior of the system, by having deep information on the system, i.e., the correct structure and behavior of the system, we can probably detect which faulty parts of the system can account for the symptom; however, only possessing knowledge derived from our experience , we can not diagnose the system unless we ever dealt with a similar symptom in our experience. For example, if from experience in dealing with a plain electric circuit as shown in Figure 1.1, we acquire only the following piece of knowledge:

if *bulb1 is blown* **then** *bulb1 does not light*,

we will not be able to face a symptom with respect to, say, *motor1*, e.g., *motor1 does not rotate.* However, if we know the structure of the device, which is constituted by *components* and *connections* among them in the device, that is to say, battery1, switch1, bulb1, motor1, and their interconnections, and understand the behavior of the device, which is composed of the behavior of each component, for example, information on how electricity and electrical voltages behave in each component, not only can we encounter the symptom, we can also sufficiently give more explanations of why the behavior of motor1 deviates from the normal behavior. Those, for example, are probably motor1 malfunctioning, another component malfunctioning, or some bad connections in the device, which cause electricity not to flow into the

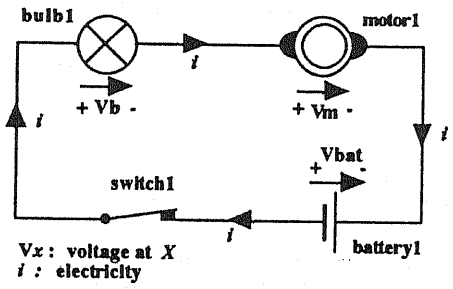


Figure 1.1: A plain electric circuit

bulb.

Therefore, the basic concept of model-based diagnosis can systematically be shown as in Figure 1.2[23, 27, 42, 44]. On the left there is the actual system (some

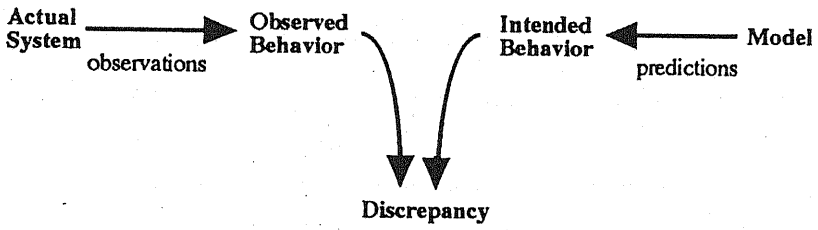


Figure 1.2: Model-based diagnosis

physical artifact) whose behavior we can observe, while on the right is a model of that system that can predict about its intended behavior. As we have known, in principle the model contains information about the correct structure and behavior of the device. We always assume that the model is always correct, so if we find any discrepancy between the observed behavior and the intended behavior, which is then considered to be a symptom, we can be sure that the discrepancy is caused by faults in the actual system, not in the model. Furthermore, we can also use the discrepancy as hints to characterize and locate the fault in the system. Since

this approach to diagnosis utilizes the model, which is a few basic principles of the system, it is also called *diagnosing from first principles.*

Now, however, in order to carry out this approach, we naturally face the following problems:

- **How do we represent the structure and behavior of a system ?** We have known that having information about the domain is useful in diagnosing, but to realize this, we require explicit representations of the structure and behavior of the domain.

- **How do we describe symptoms ?** Model-based diagnosis puts emphasis on the model. As a result, there is little information about symptoms. Therefore, we need primitives to represent them.

- **What kinds of diagnostic strategies can we apply ?** After we have explicit description of the structure and behavior of the domain, and the symptoms, then how can we do diagnosis using them ? At least, we require strategies or tactics to do this.

The aforementioned problems are necessary to be answered if we want to build a model-based diagnostic system.

The first paragraph of this section explained that in diagnosing knowing first principles was more important than having associational rules for diagnosing. However, is just knowing first principles sufficient for diagnosing ? The answer is of course no. Using only first principles, generally we only can detect what are called *behavioral faults*: faults that are caused by one or more faulty components in the device under consideration. But we can not diagnose *structural faults*: faults that are caused by structural modifications of the device, since we have only the correct structure of the device. This leads us to consider the following problem:

- **What sorts of knowledge do we really need ?** In order to detect faults that can not be detected by means of first principles, logically we need other sorts of knowledge besides first principles. We must identify those sorts of knowledge.

This problem is also worth solving in developing a diagnostic expert system.

1.4 The Goal of the Research

This research is ultimately aimed at providing a generic framework for model-based diagnostic expert systems, addressing the problems mentioned in the previous section. In developing a framework for diagnostic systems, I view the diagnostic task as a task encompassing three subtasks and exploiting three classes of information sources. The three subtasks are the subtask of generating fault hypotheses from an observed symptom *(Hypothesis Generation)*, the subtask of testing the fault hypotheses *(Hypothesis Testing)*, and the subtask of classifying the fault hypotheses that passed the test *(Hypothesis Classification)*, respectively. This division of diagnosis into the three subtasks can be regarded as an application and extension of one of basic problem-solving methods which is known in the AI literature as *Generate-and-Test Paradigm*[78]. Although it is less efficient for solving some types of problems, e.g., finding the shortest path in a road network, the paradigm gives us high lucidity and simplicity in building decision-making systems. This is why some successful AI programs such as DENDRAL[54], SOPHIE[11], and GORDIUS[86] use the paradigm as the basic strategies of their problem solvers[2]. Adding the term *Classifying* to the paradigm makes it able to be applied in building a diagnostic

[2]This does not mean that the three systems use only the pure paradigm. DENDRAL exploits *Plan-Generate-Test Paradigm* to infer the structure of organic compounds, but SOPHIE utilizes an extended *Generate-and-Test Paradigm* to construct sets of fault hypotheses, while GORDIUS applies *Generate-Test-and-Debug Paradigm* to solving problems in geologic interpretation.

system, which has a facility for final decision by providing a set of fault hypotheses classified according to a certain criterion.

The three classes of information sources are as follows. Sorts of knowledge that are useful for representing the system being diagnosed are classified into the *Domain Model.* Different from this, kinds of knowledge that are helpful in diagnosing faults, but can not directly be derived from the system are grouped into the *Additional Domain Knowledge.* The last one is a set of strategies for diagnosing, that is, the *Diagnostic Strategy.*

Therefore, it is clear that to construct the whole framework means to separately build three frameworks for the three subtasks and then to unite them into a single complete framework. Besides, we have to identify sorts of knowledge belonging to both of the domain model and the additional domain knowledge, and sorts of diagnostic strategies in each subtask. Currently, three sorts of knowledge belonging to the domain model, namely, *Device Model, Process Model,* and *Topological Relative Position* are used to model the device being diagnosed. *Heuristics*, *Naive Physics*, *Component Durability*, *Component Failure Rates*, and *Component Observability* which are classified into the additional domain knowledge, are employed as additional information to diagnose faults. As strategies for diagnosing faults, the whole framework applies three sorts of diagnostic strategies: the *Qualitative Value Propagation, Direct Path of Causality, Structural Fault Localization, Pseudo Fault Hypothesis Elimination, Contradicting Fault Hypothesis Elimination*, and *Candidate Fault Verification.* Algorithms for coordinating the sorts of knowledge and the strategies in diagnosing faults are also provided in each framework.

The framework was implemented in a computer program called MODEST and tested on the domain of refrigeration plants and evaluated using some possible faults in the domain. It showed that it could detect a structural fault that even the current

model-based approaches can hardly diagnose. Although there is still a lot of issues remaining to be done, on the whole, the framework is an important breakthrough in model-based diagnosis and should become a step towards providing a framework for diagnostic systems.

1.5 The Organization of the Thesis

The remainder of this thesis is organized in the following manner. In Chapter 2, I review some pieces of research relating to model-based diagnosis and discuss their advantages and disadvantages.

Chapter 3 concisely describes the proposed generic frame work for model-based diagnostic systems, giving a brief explanation for each subtask and each class of information source in the framework. With a theoretical and simple example, it succinctly presents the process from generating fault hypotheses to classifying fault hypotheses.

In Chapter 4, I discuss the framework for the subtask of hypothesis generation in greater detail. I elaborate on why *Device Model*, *Process Model*, and *Topological Relative Position* are instrumental in modeling the system that is to be diagnosed. In addition, I detail *Heuristics* as well as *Naive Physics* and explain why they are important in diagnosing structural faults. This chapter also clearly describes the three diagnostic strategies, namely, the *Qualitative Value Propagation*, *Direct Path of Causality*, and *Structural Fault Localization*. Furthermore, an algorithm for coordinating the sorts of knowledge and the strategies is also presented in this chapter.

In Chapter 5, I describe the framework for hypothesis testing, identifying what kinds of fault hypotheses should be pruned, how to prune them. In this chapter, I introduce *Pseudo Fault Hypotheses*, *Contradicting Fault Hypotheses*, and *Candidate*

Faults as faults that should be considered in the testing subtask.

Chapter 6 discusses hypothesis classification, giving explanations that *Component Durability*, *Component Observability*, and *Component Failure Rates* can be used as criteria for classifying fault hypotheses and together with *Binary Split Strategy* can be exploited to minimize the cost of probing.

Chapter 7 describes the implementation of MODEST, and shows how MODEST can detect a structural fault and a behavioral fault in the domain of refrigeration plant.

Finally, in Chapter 8, I summarize the key aspects of the research, discuss limitations of the proposed framework for hypothesis generation and consider directions for future work.

Chapter 2

Related Research

2.1 Introduction

To know the real current state of research on model-based diagnosis, of course, requires and wastes a great deal of time. At least we must search for all pieces of significant work on the field during the last decade. However, looking into several influential ones can help us have a general picture of it.

This chapter reviews three pieces of research relating to model-based diagnosis. At the beginning it concisely describes the three pieces of work: Davis' Constraints Suspension Approach [23], General Diagnostic Engine (GDE) [27, 29], and Knowledge Compiler II (KCII) [105], respectively. At the end it concludes the description, discussing salient advantages and drawbacks of the three approaches.

2.2 Davis' Constraint Suspension Approach

Perhaps, it is not too much to say that Davis' constraint suspension approach is an early milestone in research on model-based diagnosis. It can be regarded as the pioneer of the research; it is the first pure model-based diagnostic method.[1]

Davis, in [23], describes a model-based diagnostic method using a model of a

[1]It is pure in the sense that it uses mostly information on the structure and behavior of the device, instead of fault models.

system, represented as a set of constraints, to generate a diagnosis by testing for inconsistencies within the constraint network. These constraints depict the behavior of particular system elements, and a diagnosis is generated by removing suspects from the system model (that is to say, suspending the associated constraints) and checking for remaining inconsistencies.

The approach builds the basic level of its structure description on three concepts: *Modules, Ports*, and *Terminals*. A module can be thought of as a standard black box representing a device; ports are the places where information flows into or out of a module. Every port has at least two terminals, one terminal on the outside of the port and one or more inside. Terminal are primitives in the sense that they are the places we can probe to examine the information flowing into or out of a device through a port, but they are otherwise devoid of interesting substructure. Two or more connected devices are represented with two or more modules that are attached to each other by joining their terminals. Thus, for example, for a context in which a multiplier is connected to an adder, the structure description can be shown as in Figure 2.1. To describe substructure of a module, if any, the approach uses the

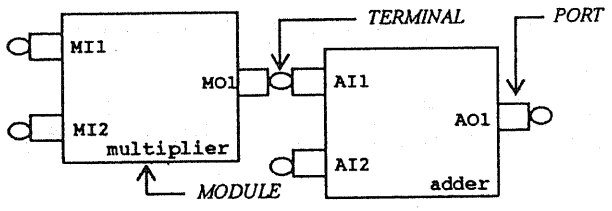


Figure 2.1: The structure description of a context in which a multiplier is connected to an adder

same concepts, i.e., the substructure is also modeled in terms of modules, ports, and terminals.

The approach applies *constraints*, which conceptually represent the relationship

among terminals on ports, to delineate the behavior of a module. For instance, as can be seen in Figure 2.2, the behavior of an adder can be grasped with three expressions as follows:

- Given the values at `I1` and `I2`, the value at `O1` is `I1 + I2`.
- Given the values at `O1` and `I1`, the value at `I2` is `O1 - I1`.
- Given the values at `O1` and `I2`, the value at `I1` is `O1 - I2`.

The behavior of another module, such as a multiplier, is also similarly explained. Thus, it is expressed by means of the logic levels of the terminals on the ports of the module.

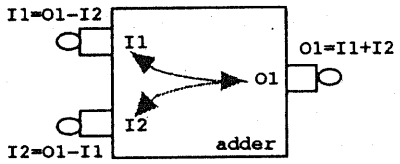


Figure 2.2: The behavior description of an adder

Using the structure and behavior descriptions mentioned above, the approach generates fault hypotheses in the following way:

- First, it simulates the device under consideration by inserting the device inputs into the constraint network that is fundamentally the model of the device, and then propagating them from a component to another component in the constraint network—this technique is known as *value propagation* or *constraint propagation*. During the simulation, every component (or module) constraint that was used to generate any particular value is maintained in a record which is called a *dependency record*. It also collects discrepancies, differences between

predicted values and observed values, or contradictory conclusions for the same point, in the constraint network during or after the simulation.

- For each discrepancy, it then generates candidate fault hypotheses (candidate faulty components) by tracing back from the point where the discrepancy occurs through the dependency records, noting each constraint involved in generating the discrepancy. To reduce the space of searching, it applies the *direct path of causality* strategy to this task. Since constraints map to components of the device, each component on which its constraint can account for the discrepancy can be regarded as a candidate fault hypothesis.

- Finally, to test the consistency of the generated candidate fault hypothesis, it *suspends* the constraint accounting for the candidate fault hypothesis, i.e., it removes the constraint from the constraint network, and then re-run the remaining constraint network. If this results in a consistent network, i.e., there is no discrepancy observed, then the candidate fault hypothesis is an explanation for the discrepancy and is accepted as a fault hypothesis. However, if the reduced constraint network is still inconsistent, i.e., there is still a discrepancy observed, then the candidate fault hypothesis is not the cause of the discrepancy. This is, of course, based on *a single failure assumption.* If multiple faults are being considered, then the candidate fault hypothesis may still be part of the cause of the discrepancy.

For example, suppose that we have an a simple logic circuit as shown in Figure 2.3 with the set of actual inputs and outputs (depicted by figures in parentheses) and the predicted outputs (represented by figures in brackets). From the figure, we know that there is a discrepancy at point F since the value of the actual output at point F is not the same as that of the predicted output at that point. This

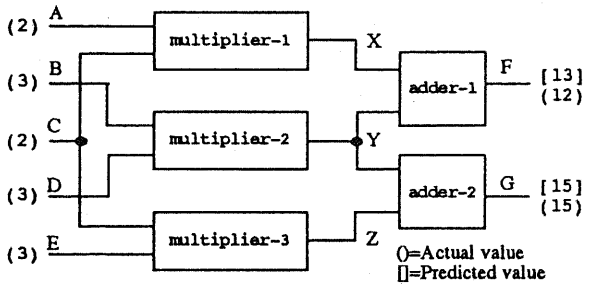


Figure 2.3: A simple logic circuit with a discrepancy at F

leads to an inconsistency of the constraint on adder-1, which results in Candidate Fault Hypothesis (henceforth CFH) 1: adder-1 is broken. Assuming adder-1 is good and tracing back from the point through the dependency records, we find a discrepancy at X or at Y, which gives us CFH2: multiplier-1 is broken, or CFH3: multiplier-2 is broken. Thus, we have three candidate fault hypotheses. Suspending the constraint on multiplier-1 or the constraint on adder-1 causes the reduced constraint network to be consistent; however, removing only the constraint on multiplier-2, we still find a discrepancy at Y. Thus, we can get rid of multiplier-2 and suspect that adder-1 or multiplier-1 is broken.

2.3 General Diagnostic Engine (GDE)

If we carefully look into the example presented in the previous section, we will find that there are still several fault hypotheses, besides adder-1 or multiplier-1, that can explain the symptom. For example, a failure in multiplier-2 together with one in adder-2 can also be considered to be a cause of the symptom. This kind of fault hypothesis is known as a *multiple fault hypothesis.*

Although Davis' approach can also generate such a fault hypothesis, that is, by

suspending two or more constraints, the combination of constraints that has to be regarded increases drastically exponentially as the number of constraints increases. Thus, Davis' approach is, practically, inefficient to generate multiple faults hypotheses.

GDE that is developed by de Kleer and Williams[27] overcomes the shortcoming by providing an efficient diagnostic procedure to generate multiple faults hypotheses. It performs a diagnosis by identifying possible faults, which are characterized as assumption violations. An assumption is the correct behavior of each component. GDE also presents a strategy for measurement selection by taking into account the fault probability of each component and the concept of minimum entropy. However, this is not a salient contribution of GDE to model-based diagnosis. It is the provided way of generating multiple fault hypotheses that is the main feature of GDE.

Although the research on GDE has already been extended by many other researchers, such as Freitag in making measurement techniques in GDE more generic[38] and in improving goal-driven structural focusing techniques in GDE[39], Struss in integrating fault models into GDE[90], and Friedrich in applying physical impossibility notions in GDE[40], the work of de Kleer and Williams' is still the most fundamental information in order to grasp how GDE works.

In order to describe the structure of the device being diagnosed, a frame-based component language based on the concept of *component-connections*, which is basically similar to the structure description of Davis' approach, is applied in GDE. And as in Davis' approach, constraints are as well exploited to represent the behavior of the device. Besides those, GDE is also outfitted with an assumption-based truth maintenance system (ATMS)[2] in order to maintain its dependency record so that it can preserve not only values but also assumptions. It is mainly this truth

[2]See [26] for details.

maintenance system that enables GDE to generate multiple fault hypotheses.

The construction of the set of fault hypotheses including multiple fault hypotheses in GDE can be divided into two stages: *conflict recognition* and *fault hypothesis generation.* These can be further explained as follows:

- **Conflict recognition**

  In this stage, *conflicts* are generated by considering an observation from which a prediction is made about the effect of, or cause of, that observation. This is done by applying the constraint propagation technique similar to that of Davis' approach. Unlike Davis' approach, however, GDE records not only the predicted values and the used constraints but also the set of assumptions underlying the predictions. If two contradictory predictions are generated for the same point in the device, then a conflict is therefore created, which in principle is the union of the assumptions underlying the predictions associated with that point. The conflict indicates that at least one of the components in it must be not working correctly.

  Generating all conflicts as explained above, however, would in the worst case be exponential. To reduce this, therefore, GDE exploits the concept of minimality, maintaining only minimal conflicts. This is also based on the intuition that any superset of a conflict is also a conflict, so it is only necessary to consider minimal conflicts.

- **Fault hypothesis generation**

  After constructing minimal conflicts, GDE comes into the phase of generating fault hypotheses. It generates a fault hypothesis for each assumption in a conflict. Since the generation of fault hypotheses can also be exponential, only minimal fault hypotheses are maintained.

The set of fault hypotheses is then incrementally modified. Thus, whenever a new minimal conflict is found, any previous minimal fault hypothesis that does not explain the new conflict is replaced by one or more superset fault hypotheses which are minimal based on this new information. This is carried out by replacing the old minimal fault hypothesis with a set of tentative new minimal fault hypotheses each of which contains the old fault hypothesis plus one assumption from the new conflict. Any tentative new fault hypothesis which is subsumed or duplicated by another is eliminated, and the remaining fault hypotheses are added to the set of new minimal fault hypotheses.

To more clearly understand the two stages, let us apply them to the example problem presented in the previous section, and see how GDE carries out the diagnosis process. As shown in Figure 2.4, assuming that `multiplier-1` is working correctly, GDE predicts that the output value of `multiplier-1` at X should be 4. It then records both the value and the set of the underlying assumption.

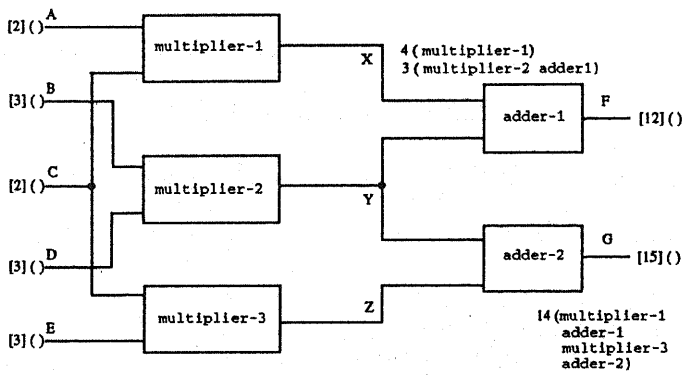


Figure 2.4: A simple logic circuit with values and assumptions generated by GDE

In the figure, assumptions are depicted with parentheses enclosing, and to distinguish predicted values from observed values , predicted values are depicted as they

are, while observed values are depicted with brackets enclosing. A predicted value and a set of assumptions underlying it are written together, by putting the later after the former. Observed values are, of course, obtained without any assumptions, so they are indicated by the null set.

GDE does the same for other components in the device. Assuming that `multiplier-2` together with `adder-1` is functioning, however, GDE predicts that the value at X should be 3. This conflicts with the previous value which is 4. Therefore, GDE generates the following conflict which is the union of the two sets of assumptions at that point:

C1: ⟨`multiplier-1 multiplier-2 adder-1`⟩

Since all the elements of the conflict are needed to detect the discrepancy at X, the conflict is a minimal conflict. Accordingly, GDE does not continue the propagation process with respect to any superset of the conflict.

In a similar manner, ignoring `multiplier-2`, and assuming that `multiplier-1`, `multiplier-3`, `adder-1`, and `adder-2` are correctly functioning, GDE calculates G to be 14. This leads to a discrepancy at G because it conflicts with the observed value which is 15. Following this, GDE then produces the second conflict:

C2: ⟨`multiplier-1 adder-1 multiplier-3 adder-2`⟩

which is a minimal conflict as well.

Those are all the minimal conflicts in the device. GDE then enters the fault hypothesis generation phase. It first generates the initial minimal fault hypotheses that account for C1; these are [`multiplier-1`], [`multiplier-2`], and [`adder-1`]. This is illustrated by the effect of C1 line in the space of fault hypotheses, which is visualized in terms of a subset-superset lattice, as shown in Figure 2.5[3]. Next, when

[3]In Figure 2.5, all elements below a line are elements that are eliminated when conflict(s) labeling the line are considered.

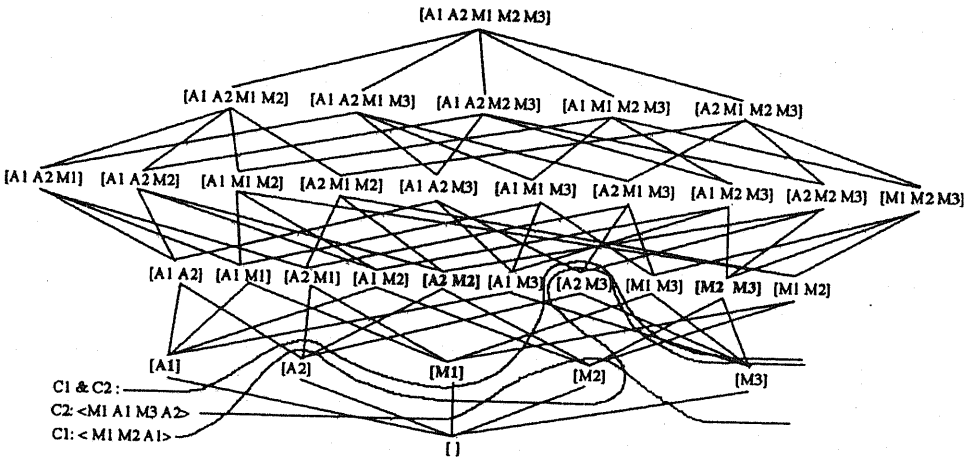


Figure 2.5: The fault hypothesis space for the logic circuit example with the effects of conflicts

considering C2, GDE gets rid of [multiplier-2] since this fault hypothesis does not explain the conflict; however, it still has to consider the immediate supersets of [multiplier-2], i.e., [multiplier-1 multiplier-2], [multiplier-2 multiplier-3], [adder-1 multiplier-2], and [adder-2 multiplier-2], because they are still probable to be fault hypotheses. Knowing that [multiplier-1 multiplier-2] and [adder-1 multiplier-2] are supersets of [multiplier-1] and [adder-1], respectively, GDE then eliminates the two supersets. Thus the new minimal fault hypotheses are [multiplier-2 multiplier-3] and [adder-2 multiplier-2]. This finally results in the following set of minimal fault hypotheses:

FH1: [adder-1]

FH2: [multiplier-1]

FH3: [multiplier-2 multiplier-3]

FH4: [adder-2 multiplier-2]

which the effect is depicted by C1 & C2 line in Figure 2.5. From this result, we can

see how cleverly GDE can generate multiple fault hypotheses.

2.4 Knowledge Compiler (KCII)

So far, I have explained two approaches to model-based diagnostic systems which mainly place emphasis on ways of generating fault hypotheses, and are merely based on information about structure and behavior of the device in the process of generation. However, there are other approaches which, rather than emphasizing information on the structure and behavior, lay stress upon what kinds of knowledge are important to generate fault hypotheses. The work on KCII developed by Yamaguchi et al. can be considered to be such a kind of approach[105]. In this section I explain the work briefly.

Although the work on KCII[105] is mainly aimed at providing a framework for acquiring heuristics for fault diagnosis based on knowledge compilation, KCII itself can be viewed as a diagnostic system. This is because, given a fault symptom, KCII generates fault hypotheses in the form of a *fault tree* which describes causal relationships between the symptom and the fault hypotheses.

Taking account of two prominent criteria for designing diagnostic expert systems: constructibility and generality, KCII proposes five kinds of knowledge that are needed for diagnosis. Those are Device World, Physical World, Control World, Interpretation World, and Failure Mechanism World. Each knowledge can succinctly be explained as follows:

- **Device World (DW).** DW constitutes the model of the device under consideration. In principle, it represents the structure and function of the device by qualitatively denoting the function of each component and parameters flows in the device. Slightly different from the structure and behavior descriptions

of the two model-based diagnostic systems described previously, DW does not describe the behavior of the component. Instead, it only describes parameters attached to the component. Relationships among the parameters are then separately described in the Physical World. Figure 2.6 shows an example description of a compressor-cylinder (a component in air conditioners) in DW.

compressor-cylinder
function:
refrigerant(outpress) = PLUS
structure:
[out(1), refrigerant(press), in(1)],
[out(2), lubricating-oil(q), in(2)],
[in(1), compressor-rotor(torque), out(1)]
parameters:
press, q, torque

Figure 2.6: A model of a compressor-cylinder

- **Physical World (PW).** PW covers physical principles or laws governing the device. It simply represents not only relationships among parameters involved in the device, but also the qualitative proportion among parameters in a constraint set or an equation. Thus, for example, in the case of Ohm's Law: $Voltage(V)/Current(I) = Resistance(R)$, by assuming that R is constant we know that if the value of V decreases the value of I increases, or vice versa. In other words, qualitative proportion explain the effect of a change in a parameter to another parameter.

- **Control World (CW).** CW includes information that can be used to control the process of generating fault hypotheses. Those, for example, are durability of components, changeability of parameters, etc. Due to this information, KCII can reduce the space of searching in generating fault hypotheses. That

is, it does not need to further analyze the fault possibility of a component if the durability of the component is high.

- **Interpretation World (IW).** KCII represents symptoms and fault hypotheses in terms of parameter values, e.g., $pipe(r) = plus$. Therefore, they are difficult to understand unless they are interpreted in terms of faults or failures. To overcome this difficulty, IW provides information relating the parameter values to the corresponding faults. Thus, $pipe(r) = plus$ can be interpreted that *the pipe is injured.*

- **Failure Mechanism World (FMW).** In diagnosing faults, there are some primary faults that are difficult to be analyzed if only based on the concept of structure and function of the device. In order to be able to detect such faults, FMW provides generic heuristics explaining how they take place. Thus, for example, *if a pipe leaks then it can be assumed that there is a chink or a hole in the pipe.* KCII uses this knowledge if it is especially insisted to do a deep analysis of a symptom.

Given a symptom, exploiting the aforementioned kinds of knowledge, KCII first propagates the symptom to the DW, and checks which component whose function is destroyed by the symptom. If there is such a component then KCII generates a fault hypothesis with respect to the component; and/otherwise, it then soon propagates the symptom to the PW and looks into parameters which receive the effect of the symptom. If possible, KCII generates fault hypotheses from these parameters. Next, KCII treats the parameters as new symptoms and propagates them backward through the parameters flows or information pathways in the device. And then, it do the same as earlier. During the process, KCII takes account of the CW and IW as well to reduce the space of searching, and to interpret fault hypotheses in terms

of faults, respectively. This process of generating fault hypotheses is also called the *retrospective reasoning phase* of KCII, which is no more than an application of the value propagation technique as exploited by the two preceding approaches .

Moreover, KCII also analyzes the interpreted fault hypotheses in order to identify primary faults. This is carried out by searching the FMW and finding out the heuristics or rules that match the fault hypotheses. KCII then use the rules to detect phenomena accounting for the fault hypotheses, and make the phenomena to be new symptoms or other fault hypotheses. This is the *pattern matching phase* of KCII.

Continuously, KCII repeats the two phases mentioned above in turn until all possible fault hypotheses are generated. Figure 2.7 shows a part of the fault tree constructed by KCII when given a symptom that *the rotation-frequency of compressor-rotor decreases (c-rotor(n) = minus).*

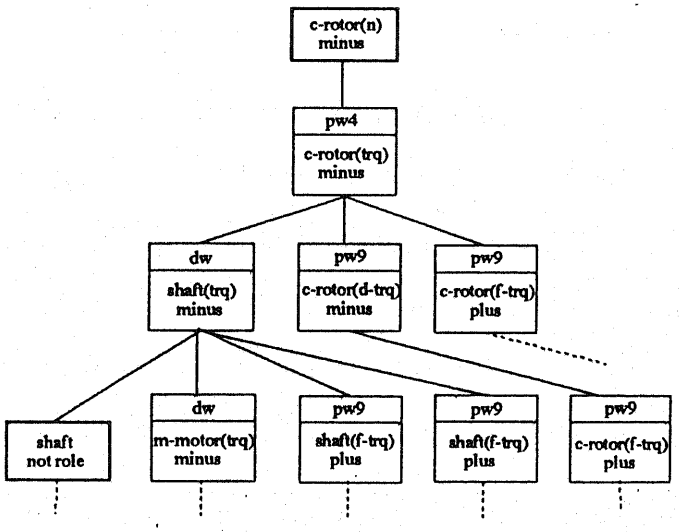


Figure 2.7: A part of a fault tree constructed by KCII

2.5 Discussion

This chapter has presented a review of three pieces of research relating to model-based diagnosis: Davis' constraint suspension approach, GDE, and KCII. As we have seen, from the point of view of what kinds of knowledge are used for diagnosis, the first two approaches can be categorized into *pure model-based diagnostic systems* since they mainly put emphasis only on the use of information on behavior and structure of a device, while the last one can be regarded as a *nonpure model-based diagnostic system* due to its use of other kinds of information besides information on behavior and structure of a device.

Davis' constraint suspension approach suggests three concepts: *Modules*, *Ports*, and *Terminals* to describe the structure of a device. These enable us to have an explicit representation of a device, which is indispensable for diagnosis. The concept of *constraints* that this approach provides facilitates us to grasp the behavior of a device, that is, we can derive behavior into mathematical equations which are logically manageable. The methods of *value propagation*, *direct path of causality*, and *constraint suspension*, which are applied in this approach, together show us a simple but convenient way to generate fault hypotheses, particularly when *a single failure assumption* is allowed.

So do those of GDE. GDE practically revises the technique of Davis' for recording inferences drawn by constraints by applying ATMS. Thus not only values can be propagated and recorded, but also assumptions underlying them can be. This enables it to deal with not only a single fault but also multiple faults.

However, if we look further into the two pure model-based diagnostic approaches mentioned above we will soon understand that both the two approaches assume that the device, or domain in general, is always a *closed-world domain*, i.e., a domain

whose all behavior can precisely be characterized in terms of constraints. Consequently, they will find difficulties if they have to deal with an *open-world domain* such as a medical domain, a mechanical domain, etc., whose detailed behavior is difficult to be grasped. The reason is that they will not have the precise model of the domain, which results in the difficulty in performing value propagation.

Besides, almost pure model-based diagnostic systems, including the two approaches above, always assume that the structure of a device does not change. They only take for granted that faults in a device are caused by one or more faulty components in the device which are showed by the deviation of parameter values and internal states—*behavioral faults.* However, those are not always like that. The structure of a device can be subject to changes, and faults can be caused by those changing device structure—*structural faults.* Since the mechanisms of generating fault hypotheses in pure model-based diagnostic systems are based only on the correct behavior and structure of a device, it is difficult, or even impossible, for pure model-based diagnostic systems to face with such faults.

Nonpure model-based diagnostic systems attempt to solve the above problems by considering other sources of information besides one on the structure and behavior of a device. KCII does this by introducing IW, CW, and FMW. Among the introduced kinds of knowledge, FMW can be considered to be the most prominent since it includes primitive heuristics explaining how particular faults take place. Due to this kind of knowledge, KCII can detect faults that are difficult to find out only with value propagation. Although this partly solves the first problem, however, it does not give a solution to the second problem yet. We still need other sorts of knowledge beyond heuristics to address the problem.

Besides the aforementioned limitations, the three approaches reviewed in this chapter possess other limitations on pliability. It is difficult to integrate additional

knowledge and strategies with the preset knowledge because there are no clear classification on types of information sources, and explicit division on the task of diagnosis. In the following chapter, I discuss these issues and attempt to provide a unifying framework for dealing with all those limitations.

Chapter 3

An Extended Model-Based Diagnosis Paradigm

3.1 Introduction

In the introduction of this thesis, it has been stated that diagnosis is the process of identifying possible faults from some observed symptoms. To make a program performing this process is difficult unless we have the clear understanding of what kinds of simpler tasks make up the process and what kinds of information sources can be used to carry out the process.

Much work in model-based diagnosis, including the first two pieces of work described in the previous chapter, has indicated, although not explicitly, that *to diagnose* is *to generate*, *test*, and *discriminate* fault hypotheses. However, they clearly provide neither a framework for each task nor information sources for performing each task.

In this chapter, extending the basic paradigm I explain that the task of diagnosis can be viewed as a task encompassing three subtasks: *Hypothesis Generation*, *Hypothesis Testing*, and *Hypothesis Classification*, which exploit three information sources: *Domain Model*, *Additional Domain Knowledge*, and *Diagnostic Strategy*, as can be seen in Figure 3.1. In the figure, bubbles in each domain of the information

sources denote elements of each information source, i.e., sorts of knowledge involved there. I first concisely describe the three information sources, and then outline the three subtasks, depicting the process from generating to classifying fault hypotheses.

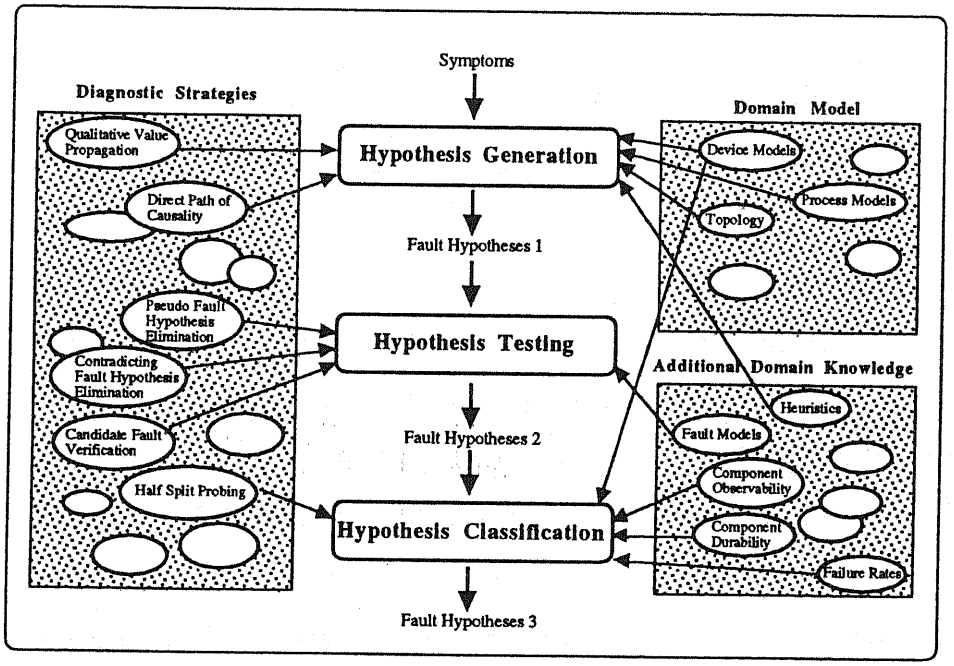


Figure 3.1: An extended model-based diagnosis paradigm

3.2 Three Information Sources for Diagnosis

When trying to solve a certain problem, i.e., performing a task of reasoning, we usually start by asking ourselves a question of what kind of information can be used. Conventionally, AI literature addressed this problem by suggesting the importance of the notions of *knowledge representation* and *problem-solving methods* [78]. It, however, rarely pointed out what classes of knowledge or information are really required for performing the task. Although the notions of knowledge representation and problem-solving methods are important in systematizing the task, i.e., in making

a computer program carry out the task, I think that we should know first about those classes before coming into the notions themselves.

Considering useful sorts of knowledge and strategies for diagnosing faults, I divide information sources for diagnosis into three classes. One is a class of knowledge that is instrumental in representing the domain under consideration (Domain Model), another is a class of knowledge that is helpful in diagnosing faults but can not be derived directly from the domain itself (Additional Domain Knowledge), and the other is a set of strategies for diagnosis (Diagnostic Strategy). Each of them can be briefly explained as follows:

- **Domain Model.** The domain model is fundamentally a class of knowledge that is provided by the domain itself; thus, it features the nature of the domain. If the domain is a device it describes, for example, the device structure, the device behavior, the device function, actions in the device, the device topology, and so on. Thus it can be regarded as the main information sources of model-based diagnosis. As can be seen in Figure 3.1, sorts of knowledge belonging to this class, for example, are the device model, process model, and topological relative position. These particularly will be explained further in the next chapter.

- **Additional Domain Knowledge.** However, to be able to diagnose faults in a device often requires other sorts of knowledge lying outside the device. That is to say, the domain model itself sometimes is not adequate for performing the task. This is what we actually see and feel in our daily live. For example, in case of diagnosing a TV, although the design description of the TV is a fundamental information for performing the task, only knowing it we still often find diagnosing the TV difficult. We still need other kinds of information

such as fault models, component observability, component durability, etc. in order to be able to perform the task. The class of knowledge that covers such kinds of information is what I term the additional domain knowledge.

- **Diagnostic Strategy.** After we have the two classes of knowledge above, the left problem that should be addressed is how to employ them in carrying out the task of diagnosis. This is related to the problem of identifying problem-solving methods in diagnosing faults, involving problems of how to detect structural faults as well as behavioral faults. In other words, we have to identify strategies for diagnosing them. Diagnostic strategy refers to those strategies, indicating sorts of information related to problem-solving methods for diagnosis. Several examples of those sorts of information can be given as follows: qualitative value propagation, direct path of causality, structural fault localization, constraint suspension, etc. (See Figure 3.1). Recalling the previous chapter, we know that the approach as well as methods used by Davis in [23] is categorized into a diagnostic strategy in the current paradigm. In the next chapter, the first three of the aforementioned strategies are explained in greater detail.

Having had all the three information sources above, what we should do next is arrange clear steps to reach the goal of diagnosis. This is the topic of the next section.

3.3 Three Subtasks of Diagnosis

Decomposing a task into several simpler subtasks is one way to reduce the degree of difficulty in accomplishing the task. Davis and Hamscher stated that diagnosing faults could be viewed as a sequence of three subtasks: generating, testing, and dis-

criminating fault hypotheses [24]. It is this that is well known as the basic paradigm of model-based diagnosis. Modifying this, I similarly consider that diagnosing faults is performing the tasks of generating fault hypotheses (Hypothesis Generation), testing fault hypotheses (Hypothesis Testing), and classifying fault hypotheses (Hypothesis Classification), respectively. The difference between my proposed paradigm and the basic paradigm lies on the third subtask, where rather than distinguishing among competing fault hypotheses, I put emphasis on classifying fault hypotheses according to some certain criteria. The following concisely describes each subtask:

- **Hypothesis Generation.** The basic task is, given a symptom, generate fault hypotheses. In this stage, a hypothesis generator appropriately employs sorts of knowledge and/or strategies available in the three information sources in order to determine possible fault hypotheses from a given symptom (See Figure 3.1). Thus, for example, in diagnosing faults in *Device A*, say, it produces the following set of possible fault hypotheses (FHs):

$$FHs : \{fh1, fh2, fh3, fh4, fh5, fh6\}$$

where fhi indicates a possible fault hypothesis.

As can be studied from AI literature, e.g., [104], it is obvious that the hypothesis generator should be *complete*, that is, it is capable of producing all possible fault hypotheses; *nonredundant*, in the sense that it never generates the same fault hypothesis twice; and *informed*, that is, it uses possibility-limiting information, restricting itself to producing few fault hypotheses that finally prove to be inconsistent. In attempting to build a framework for this task, I follow those properties as possible as I can, letting the hypothesis generator produce all possible fault hypotheses or setting the number of fault hypotheses the

hypothesis generator should achieve, checking each generated fault hypothesis to see if it is really generated ones only, and making the hypothesis generator employ sorts of knowledge and strategies as appropriately as it can. The next chapter discusses the framework in detail.

- **Hypothesis Testing.** Because it is natural that all sorts of provided knowledge and strategies can not capture all real information on or around the device, the hypothesis generator can not avoid producing some inconsistent fault hypotheses. Therefore, in order to get plausible fault hypotheses, we have to check all generated fault hypotheses and find such inconsistent ones and then discard them. This task is done by a hypothesis tester in hypothesis testing, by properly exploiting available sorts of knowledge and strategies in the three information sources. The hypothesis tester will apply, for example, the technique of *Candidate Fault Verification* in examining if the state caused by a fault hypothesis fits the state shown by the symptom (See Figure 3.1). Thus following the example given in the hypothesis generation, the hypothesis tester will test the set $\{fh1, fh2, fh3, fh4, fh5, fh6\}$ and, for example, refine the set to be the following set of plausible fault hypotheses (FHs'):

$$FHs' : \{fh1, fh3, fh4, fh6\}$$

where fhi indicates a plausible fault hypothesis.

- **Hypothesis Classification.** After we know all plausible fault hypotheses, what we still usually require is information that can guide us to make proper measurements on those fault hypotheses in order to find out the real fault quickly. In other words, we have to distinguish among those fault hypotheses by gathering new additional information. This is actually not the task of a di-

agnostic system, but we usually agree that a good diagnostic system should do the task as well. To deal with the task, conventional ways (e.g., [12, 83]) usually provide methods for either probing, i.e., making additional observations, or testing, i.e., changing the inputs and making observations in that new situation. Owing to the distinguishing of fault hypotheses, the task is also known as *Hypothesis Discrimination* in the basic paradigm of model-based diagnosis.

I, however, think that what a diagnostic system should do in the final task is classify plausible fault hypotheses into different ranked orders based on different certain criteria as needed. The ranked order itself then can be represented in terms of *a decision tree.* Those criteria, for example, are component observability, component durability, component failure rates, etc. (See Figure 3.1). Thus, following the example in the previous subtasks, if we have two criteria, say, α and β, the hypothesis classifier of the diagnostic system will produce, for example, the following two different ranked order sets (FHs''), each of which is composed of elements of the plausible fault hypotheses set generated in the hypothesis testing.

$$\alpha : FHs'' : \{fh1, fh4, fh3, fh6\}$$
$$\beta : FHs'' : \{fh6, fh3, fh1, fh4\}$$

In addition, the hypothesis classifier of course can choose only a single criterion as needed and produce the decision tree based on it to provide information for making proper measurements.

3.4 Conclusions

In this chapter, I have described the outline of the extended model-based diagnosis paradigm that I proposed, briefly explaining what kinds of information sources and

what kinds of tasks are useful for performing the task of diagnosis. The paradigm clearly divides the task of diagnosis into three ordered subtasks: *Hypothesis Generation*, *Hypothesis Testing*, and *Hypothesis Classification*. It also accurately separates diagnostic strategies from knowledge for diagnosis, classifying the information required for diagnosis into three classes: the *Domain Model*, *Additional Domain Knowledge*, and *Diagnostic Strategy*. Due to these various sorts of information, the paradigm can be considered to be a nonpure model-based diagnostic approach.

Now, the important jobs that must be done are identify bubbles in each domain of the three information sources and provide a framework for each subtask. The next chapter discusses a framework for the first subtask—hypothesis generation, elaborating on kinds of knowledge and strategies required to carry out the subtask.

Chapter 4

Hypothesis Generation

4.1 Introduction

In order to generate fault hypotheses, it can be considered that experts use their commonsense, knowledge they have gotten from their experience before, knowledge about the domain under consideration, and strategies they have known. Although how they coordinate, organize, and represent such kinds of information in their minds in carrying through the task is still obscure, identifying useful sorts knowledge and strategies for diagnosis can be started by looking into those kinds of information they use.

In this chapter, in order to provide a framework for hypothesis generation, I attempt to identify sorts of knowledge and strategies that are useful for generating fault hypotheses, and furnish an algorithm for coordinating them as well.

4.2 Overview

The proposed framework for hypothesis generation is shown in Figure 4.1. To generate fault hypotheses, the framework considers three sorts of knowledge belonging to the domain model: *Device Model*, *Process Model*, and *Topological Relative Position*. Besides, it employs *Heuristics* and *Naive Physics* as the additional domain

knowledge. For the time being, as strategies for diagnosis, the framework applies two common strategies: *Qualitative Value Propagation* and *Direct Path of Causality*, particularly to detect behavioral faults, and uses a *Structural Fault Localization* strategy to diagnose structural faults. Furthermore, the framework is also equipped

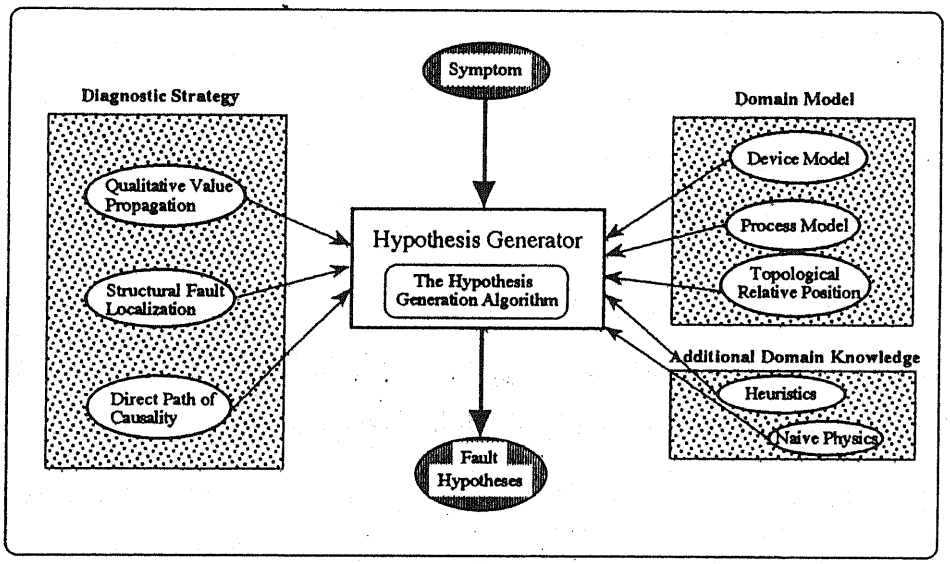


Figure 4.1: Hypothesis generation

with an algorithm for coordinating the aforementioned sorts of knowledge and strategies.

Briefly, given a symptom, the hypothesis generator of the framework, by following the hypothesis generation algorithm and using each knowledge sort or strategy of the three information sources as needed, generates fault hypotheses. In the following sections, I explain the knowledge sorts, strategies, and algorithm in greater detail.

4.3 Domain Model

Sorts of knowledge that are used directly to model or to represent a device are classified into the domain model. This section discusses those sorts of knowledge.

4.3.1 Device Model

The device model, which is the back bone of the current model-based diagnosis, plays an important role in diagnosing faults based on the structure and behavior of the device. It is due to the model that the qualitative value propagation and direct path of causality strategies can be applied in the diagnosis. In principle, it represents the structure and behavior of a component in the device. However, for the purpose of diagnosis, another information such as the purpose of the component being modeled is usually added into the model. In the framework, the device model is represented in the form of a frame that includes the following information:

- A description of the *purpose* of the component being modeled, which is represented as a set of relations among variables and qualitative values describing the final state that has to be achieved by the component.
- A set of *processes* that are active when the component works; each of the processes is then further described in the process model.
- A set of components that are *connected* to the component.
- A list of *parameters* that are used for interaction between the component and the other components connected to the component.
- A set of *constraints* describing the behavior of the component in terms of qualitative derivative equations.

Therefore, by means of the device model we can model a compressor in a refrigeration plant as shown in Figure 4.2.

Note that the device model differs from those of [23, 25, 31, 89, 91, 95] where information on the processes being active when the component works is not included.

compressor	
purpose:	
state 1:	$outpress(gas) = +$
processes:	$compression(gas, compressor)$
terminal:	
input 1:	
connection:	*conduit* 4
parameters:	$inflow(gas), inpress(gas), intemp(gas)$
output 1:	
connection:	*conduit* 1
parameters:	$outflow(gas), outpress(gas), outtemp(gas)$
constraints:	
state 1:	$[inpress(gas)] + [addpress] = [outpress(gas)]$
	$[addpress] = POS\_CONST$
	$[inpress(gas)] = M_+[intemp(gas)]$
	$[outpress(gas)] = M_+[outtemp(gas)]$
	$[inflow(gas)] = M_+[outflow(gas)]$

Figure 4.2: A device model of a compressor

I add the information on the processes to the device model because the processes can be used to explain why the device does not achieve its expected purpose.

4.3.2 Process Model

Besides being able to be represented in terms of structure and behavior, a system can be described in terms of the processes being active in the system, i.e., actions done in the system. In the process model, I model processes that are active in the system under consideration. As Forbus defined a process [37], I describe a process in the form of a frame that is specified by the following four slots:

- *individuals* : a set of objects that participate in the process.
- *initiating conditions* : a set of conditions which must hold for the process to start.

- *sustaining conditions* : a set of conditions which must hold for the process to continue.

- *effects* : a set of value parameters which are influenced by the process.

Figure 4.3 shows a process model for gas compression in the compressor. It describes that the process gas compression in the compressor will be active if there is gas in the compressor, and if the compressor gives an additional pressure to the gas. The process itself finally causes the pressure of the gas to increase.

compression(gas,compressor)	
individuals:	$gas, compressor$
initiating conditions:	$exist(gas, compressor)$
sustaining conditions:	$addpress(compressor) = +$
effects:	$press(gas) = +$

Figure 4.3: A process model of gas compression in a compressor

The first three slots of the process model are useful in generating fault hypotheses. For example, knowing that process gas compression in the compressor is always active in the normal state, so if we find the process not active, we can give the following best explanations for it as the fault hypotheses.

- by negating the individuals:

  Perhaps the gas or the compressor disappears.

- by negating the initiating conditions:

  Perhaps the gas has flowed out of the compressor, thus there is no gas in the compressor now.

- by negating the sustaining conditions:

Perhaps there is a malfunction in the compressor, so the compressor can not give an additional pressure to the gas.

Furthermore, a process can also be regarded as something that causes objects to change [37, 21]. Therefore, it can also be used to detect structural faults since structural faults are caused by changes in the structure of a device. The framework also takes into account processes as fundamental components to diagnose structural faults, using them to account for phenomena derived from the Naive Physics. This is discussed in subsection 4.4.2.

4.3.3 Topological Relative Position

In diagnosing faults, often we need the topological description of the device under consideration. The topological relative position presents the information, describing the relations among components in the device in terms of space. This kind of information is useful for controlling combinatorial explosions in generating fault hypotheses, particularly structural faults. For example, suppose there are hot thing A and cold thing B in a room, and no directly connection between them (assume a connection is something that can be seen). Knowing that A and B are in the same room, if we find the temperature of B increasing, we can deduce that the heat flowing to B is probably from A, not from the outside of the room. Figure 4.4 shows a very primitive topological relative position in the domain of refrigeration plants. It says that the compressor, propeller-fan, conduit 1, conduit 2, conduit 4 are in the same space or room that is called *outdoor-unit*.

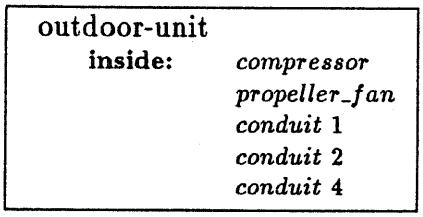


Figure 4.4: A topological relative position of an *outdoor-unit*

4.4 Additional Domain Knowledge

So far, I have explained three different sorts of knowledge: the device model, process model, and topological relative position, which are useful in modeling a device. In this section, I describe two sorts of knowledge that do not represent the structure and behavior, but are helpful in diagnosing faults.

4.4.1 Heuristics

In fact, the first task the hypothesis generator has to do, to start generating fault hypotheses, is to interpret the given symptom. Most of approaches to model-based diagnosis take an assumption that symptoms can be interpreted directly in terms of parameters values. Thus, in the case of the symptom that the value of the gas pressure in the output terminal of the compressor is less than the normal value that is +, it can be interpreted in the form: $outpress(gas) = -$. However, in fact, not all observed symptoms can be interpreted in terms of parameter values. For example, in the case of the compressor knocking, it is difficult to represent the symptom in terms of parameter values. In order to be able to interpret such a symptom, at least a certain extent of diagnostic knowledge on the domain is needed. In other words, we have to use heuristics to cope with such a difficulty. For instance, the following states that knocking at the compressor can be caused by liquid compression at the

compressor:

$$knocking(compressor) \Leftarrow compression(liquid,\ compressor)$$

4.4.2 Naive Physics

From the style of reasoning, diagnosis can also be considered, although not absolutely, postdiction. It means that the task is to deduce how the state shown by a given symptom might have happened. To do the task often requires physical common sense or naive physics, which is not available in the structure and behavior of the device under consideration. I use the following form of axioms (taxonomy), which is similar to that of [46], to formalize naive physics:

$$\begin{aligned} Q(x) &\Leftarrow Q_1(x) \vee Q_2(x) \vee \cdots \vee Q_n(x) \\ Q_1(x) &\Rightarrow T_1 \\ Q_2(x) &\Rightarrow T_2 \\ &\vdots \\ Q_n(x) &\Rightarrow T_n \end{aligned}$$

where $Q_i(x)$ are possible states or categories of x and T_i are theories of x. As the theories, processes as specified in subsection 4.3.2 are used. However, different from the processes in the process model, the processes used here are not processes being active in the system but processes explaining how an observed state might have come about. Figure 4.5 shows taxonomies of "exist" histories of states of matter. From the figure, therefore, if we find that there is a liquid at X we can deduce that perhaps the liquid is the result of a liquid *flowing* from a source (*Src*) to X, a solid *melting* at X, or a gas *condensing* at X.

At a glance, it seems that the naive physics is the same as the heuristics described in the previous subsection. But, actually they are different from each other.

Taxonomy of "exist" histories of states of matter

liquid:

$$exist(liquid, X) \Leftarrow flowing(liquid, Src, X) \vee melting(solid, X) \vee condensation(gas, X)$$

gas:

$$exist(gas, X) \Leftarrow flowing(gas, Src, X) \vee sublimation(solid, X) \vee evaporation(liquid, X)$$

solid:

$$exist(solid, X) \Leftarrow moving(solid, Src, X) \vee freezing(liquid, X) \vee condensation(gas, X)$$

Figure 4.5: A sample of naive physics

Unlike the heuristics, which is a particular diagnostic knowledge of a domain that is obtained by experience, the naive physics is our common sense about the everyday physical world, which does not require a specific study of a domain in principle.

It is mainly due to the naive physics that structural faults can be diagnosed. This will be clear in an example presented in the next chapter.

4.5 Strategy

Knowing only knowledge on the domain, both the domain model and the additional domain knowledge, is not enough to generate fault hypotheses. Besides the two classes of knowledge, we need strategies to do the task. This section describes briefly three basic strategies for diagnosing faults or generating fault hypotheses in particular.

4.5.1 The Qualitative Value Propagation and Direct Path of Causality

Two common strategies known widely in the current model-based diagnosis are the qualitative value propagation and the direct path of causality. The former states that to generate fault hypotheses, particularly behavioral fault hypotheses, in a device, given an observed symptom, it is important to interpret the observed symptom which usually can be interpreted in the level of a parameter value as a qualitative value rather than as a quantitative (numerical) one, then propagate the qualitative value through the whole device model of the device, using information on the terminal slot and constraints slot. The qualitative value itself can be distinguished into three types, namely, increasing (+), decreasing (-), and normal (0) [25]. The use of such types of values in generating faults is mainly due to the following reasons[10, 28, 92, 99]:

- Quantitative analysis can be considered impractical (i.e., it requires more computation, which actually is not necessary, than qualitative analysis).
- Representing symptoms and faults in terms of qualitative values is more plausible than representing them in terms of quantitative values.
- Sometimes, even numerical values for parameters characterizing faults and symptoms are not precisely known.

In my framework, in order to cover a wider value range, I extend these three types into seven types:

Extremely Increasing	(+ + +),
Fairly Increasing	(+ +),
Relatively Increasing	(+),

Normal	(0),
Relatively Decreasing	(-),
Fairly Decreasing	(- -),
Extremely Decreasing	(- - -).

The reason is that in some domains, a parameter whose value relatively increases, fairly increases, and extremely increases, respectively, cause different faults. The condition is the same when the value of the parameter decreases. Thus, we need different symbols to represent each qualitative state of the parameter.

However, as a device gets more complex, the qualitative value propagation becomes more to be done, which leads fault hypothesis generation to become computationally intractable. In order to overcome this problem, the direct path of causality strategy can be applied in generating fault hypotheses. Thus, it is enough to propagate an observed symptom in the opposite direction of the information flow in the device, i.e., from the component where a symptom is observed towards components that are upstream of the component.

To give an explicit example, assume that in a simple device consisting of component A, B, C, and D as shown in Figure 4.6, we observe that C looks abnormal showing a symptom that the current *flow* in C increases ([flow,+]). We can then qualitatively propagate this symptom upstream the component. Therefore, we can predict that A, B, or D is likely to be faulty.

4.5.2 The Structural Fault Localization

Owing to a certain fault, two or more components in a device that are not connected directly with each other can be unexpectedly connected with each other. Applying only the two strategies described in the previous subsection, we can not detect the so-called structural fault. This is because the structural modifications of the device

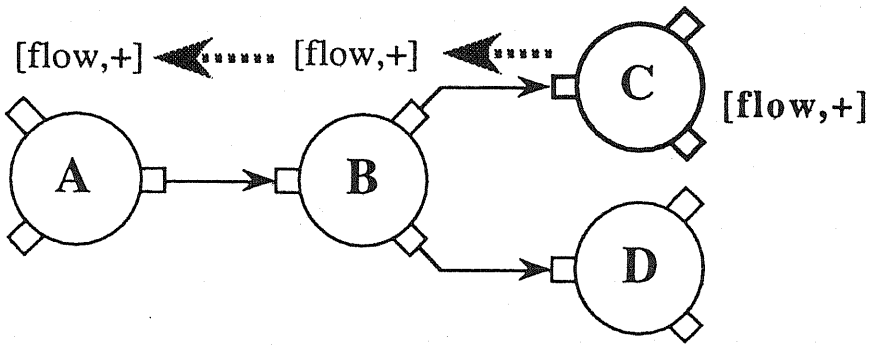


Figure 4.6: Qualitative Value Propagation & Direct Path of Causality

also cause changes to the path of causality in the device. As a way to solve the problem, the framework makes use of the naive physics, the topological relative position, and the device model to detect the fault from the effect of the fault, i.e., the observed symptom. The detection can be explained as follows. First, the hypothesis generator uses the naive physics to deduce how the state shown by the symptom might have come about; if necessary, it uses the heuristics. Next, it uses information on the corresponding processes, i.e., individuals, initiating conditions, and sustaining conditions of the processes, to detect possible faults. Finally, using the topological relative position of the device the hypothesis generator screens the possible faults then localizes possible structural faults with the device model of the device. This is the structural fault localization strategy.

As an illustration, assume that we are given a similar device with a similar state like the previous example (See Figure 4.7). However, in this case, besides component A, B, C, and D, there is component E which is not directly connected to C but whose the same *flow* and is in the same room. Therefore, if we observe that C looks abnormal showing a symptom that the current *flow* in C increases ([flow,+]), from the current situation and from the commonsense that the increasing of the current *flow* in one place can be caused by the flowing of the *flow* from another place to the place, besides propagating the value of the current flow upstream C, we

can also predict that the increasing of the current flow in C is possibly caused by the *flow* flowing from E.

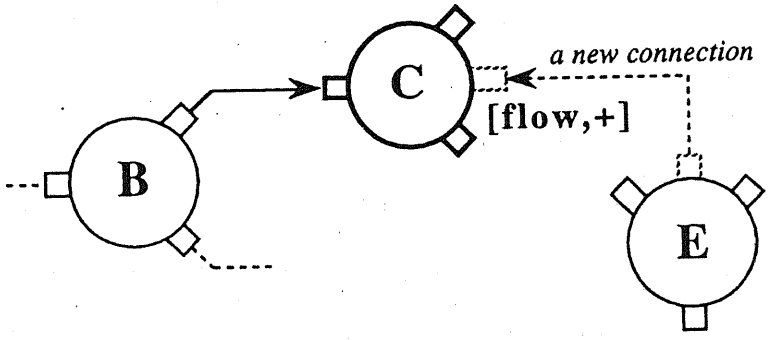


Figure 4.7: Structural Fault Localization

A more explicit example of diagnosing a structural fault in the domain of refrigeration plants that will be given in Chapter 7 will make the strategy clearer.

4.6 Hypothesis Generation Algorithm

The mechanism of generating fault hypotheses that exploits the sorts of knowledge and the strategies described in the previous subsections is shown schematically in Figure 4.8. In order to be able to use the sorts of knowledge directly in generating fault hypotheses, the framework classifies symptoms into four types, namely, parameter value symptoms, process symptoms, heuristic symptoms, and naive physics symptoms. Parameter value symptoms are symptoms that are represented at the level of parameter values. For example, the symptom that says that the gas pressure in the output terminal of the compressor decreases can be represented with $outpress(compressor) = -$. The other types of symptoms can be defined similarly; thus, process symptoms, heuristic symptoms, and naive physics symptoms are symptoms at the level of processes, the level of heuristics, and the level of naive physics, respectively. Table 4.1 gives an example of each symptom type.

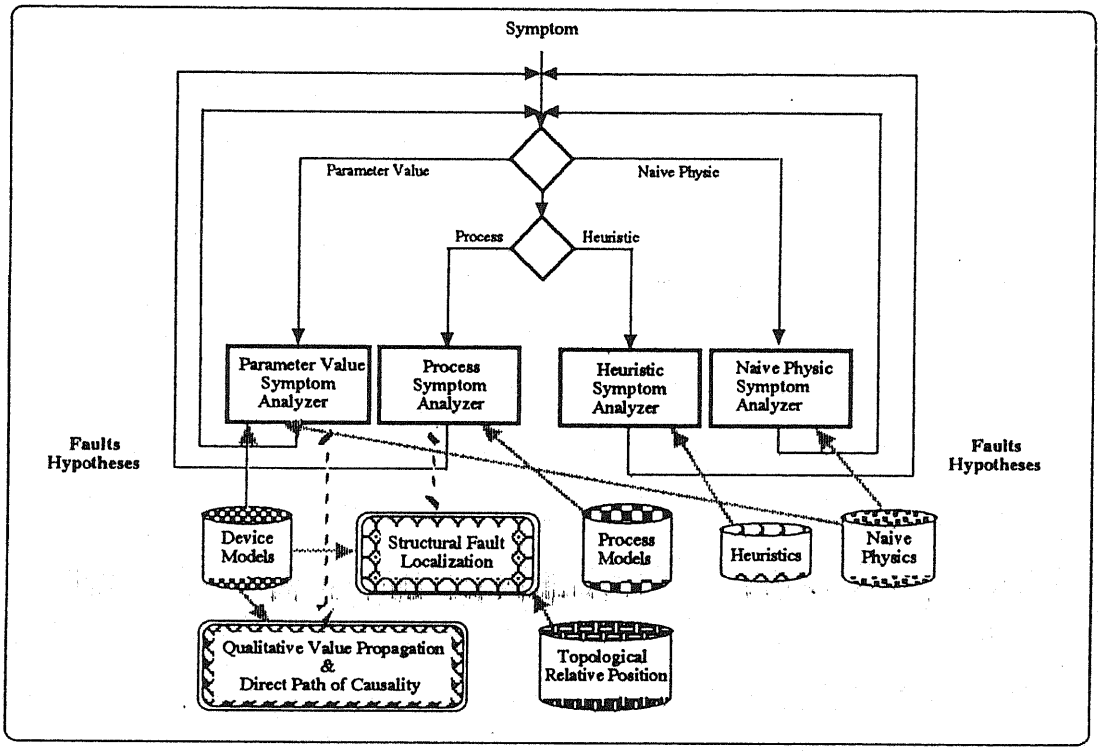


Figure 4.8: Hypothesis generation mechanism

Table 4.1: Examples of symptoms

Symptom Type	*Example*
Parameter Value Symptom	outpress(compressor) = –
Process Symptom	compression(liquid,compressor)
Heuristic Symptom	knocking(compressor)
Naive Physic Symptom	exist(liquid,compressor)

The mechanism consists of four different symptom analyzers, each of which corresponds to each of analyzers for the four types of symptoms. Given a symptom, using appropriately the five sorts of knowledge and the three strategies if needed, the mechanism generates fault hypotheses which can be derived directly from the symptom and then takes the fault hypotheses as new symptoms to generate further fault hypotheses. This is done repeatedly until all possible fault hypotheses are generated or a certain approximate number of fault hypotheses are achieved. The hypothesis generator is built based on the mechanism, and the more detailed algorithm of it is presented in Fig. 4.9. The main job of the hypothesis generator is shown in the main loop contained within the `REPEAT-UNTIL` construct. The hypothesis generator begins by picking up a symptom from the `symptom list`, and then analyzes the symptom with the corresponding analyzer. For instance, if the symptom is at the level of parameter values then the parameter value symptom analyzer will analyze the symptom. In each analyzer, the basic tasks are then `GENERATE` possible fault hypotheses which can be directly derived from the symptom, `CREATE` new different symptoms using the available sorts of knowledge, and `INSERT` the new different symptoms into the symptom list.

In performing the job, however, the hypothesis generator sometimes faces two special cases. The first one is when the hypothesis generator finds that the symptom is not the device purpose parameter but the device parameter, that is to say, one parameter included in the slot *constraints* of a device model is abnormal either increasing or decreasing. In this case, the hypothesis generator `CALL`s routine `QvpropDirpath`, which is the implementation of the qualitative value propagation and direct path of causality strategies, to generate fault hypotheses. As can be seen from Figure 4.10, in addition to carrying out the three basic operations (i.e., `GENERATE`, `CREATE`, and `INSERT`), the routine `PROPAGATE`s the symptoms through

```
INSERT   a symptom into the symptom list.
REPEAT
   GET   a symptom from the symptom list.

{Parameter Value Symptom Analyzer}

   IF   the symptom is at the parameter level
   THEN IF   the symptom is not the device purpose parameter but the device  parameter
        THEN CALL "QvpropDirpath(the symptom)"
        ELSE IF   the symptom is not the device purpose parameter and is not the device parameter
             THEN GENERATE  behavioral fault hypotheses  directly  corresponding  to  the  symptom,  if
                            possible.
                  CHECK     the taxonomy table to see what processes cause the symptom.
                  CREATE    new symptoms by using the processes.
                  INSERT    the new symptoms into the symptom list.
        ELSE IF   the symptom is the device purpose parameter
             THEN IF   the parameter value > the normal value
                  THEN GENERATE     behavioral fault hypotheses directly corresponding to the
                                    symptom, if possible.
                       EXAGGERATE   the process of the device model, modifying the sustaining
                                    condition of the process.
                       CREATE       new symptoms from the modified sustainig conditions of the process.
                       INSERT       the new symptoms into the symptom list.
             ELSE IF   the parameter value < the normal value
                  THEN GENERATE     behavioral fault hypotheses directly corresponding to the  symptom,
                                    if possible.
                       DECREASE     the process of the device model, modifying the individuals or
                                    initiating conditions or sustaining conditions of the process.
                       CREATE       new symptoms from the modified slots of theprocess.
                       INSERT       the new symptoms into the symptom list.
   ELSE

{Heuristic Symptom Analyzer}

   IF   the symptom is at the heuristic level
   THEN GENERATE   behavioral fault hypotheses directly corresponding to the symptom, if possible.
        CREATE     new symptoms by using the heuristics.
        INSERT     the new symptoms into the symptom list.
   ELSE

{Process Symptom Analyzer}

   IF   the symptom is at the process level
   THEN GENERATE   behavioral fault hypotheses directly corresponding to the symptom, if possible.
        IF   the symptom (or the process) also occurs at the corresponding device in the  normal  state
        THEN EXAGGERATE   the process of the device model, modifying the sustaining conditions  of  the
                          process.
             CREATE       new symptoms from the modified sustainig conditions.
             INSERT       the new symptoms into the symptom list.
        ELSE IF   the process does not occur at the corresponding device in the normal state
             THEN CALL "StrucFaultLoc(the symptom)" to generate structural faults, if possible.
   ELSE

{Naive Physics Symptom Analyzer}

   IF   the symptom is at the naive physic level
   THEN GENERATE   behavioral fault hypotheses directly corresponding to the symptom, if possible.
        CHECK      the taxonomy table to see what processes cause the symptom.
        CREATE     new symptoms by using the processes.
        INSERT     the new symptoms into the symptom list.

UNTIL the symptom list is empty or a certain approximate number of fault hypotheses are achieved.
```

Figure 4.9: The hypothesis generation algorithm

the upstream components connected to the device under consideration to enable the generation of other fault hypotheses.

```
"QvpropDirpath(Symptom)"
   GENERATE    behavioral fault hypotheses directly corresponding to the symptom, if possible.
   PROPAGATE   the symptom through the (upstream) components connected to the device.
   CREATE      new symptoms.
   INSERT      the new symptoms into the symptom list.
   RETURN

"StrucFaultLoc(Symptom)"
   SCAN        the symptom(i.e., the process)'s slots.
   GENERATE    structural faults hypotheses corresponding to the process's slots  by  searching
               the corresponding topological relative position and device model.
   CREATE      new symptoms from the structural faults, if possible.
   INSERT      the new symptoms into the symptom list.
   RETURN
```

Figure 4.10: The qualitative value propagation & direct path of causality, and structural fault localization procedures

The second one is when the hypothesis generator finds that the symptom is a process that normally does not occur at the corresponding device. If this happens, `StrucFaultLoc`, a procedure implementing the structural fault localization strategy, will be invoked to detect possible structural faults that have caused the process to be active. As shown in Fig. 4.10, the procedure begins by SCANning the slots of the corresponding process followed by GENERATIng possible structural faults using information on the slots along with one on related device models and topological relative positions.

Once no symptom is found in the symptom list, or fault hypotheses have been produced as many as stated by the preset approximate number, the hypothesis generator stops analyzing symptoms.

4.7 Discussion

This chapter has described a framework for hypothesis generation, identifying useful sorts of knowledge and strategies for generating fault hypotheses. It has suggested the *Device Model*, *Process Model*, *Topological Relative Position* to represent the device under consideration, and the *Heuristics* and *Naive Physics* to acquire sources of knowledge lying outside the domain model. It has also briefly indicated that the two common strategies for diagnosis, that is, the *Qualitative Value Propagation* and *Direct Path of Causality*, are sufficient in order to diagnose behavioral faults, but should be incorporated with the *Structural Fault Localization* if subjects to diagnose are structural faults. Furthermore, in order to use the sorts of knowledge efficiently, the framework has classified symptoms into four types: *Parameter Value Symptoms*, *Process Symptoms*, *Heuristic Symptoms*, and *Naive Physics Symptoms*. Although not complete, these can also be considered to be an attempt to provide real descriptions of symptoms.

The work of extending model-based diagnosis in order to cope with structural faults has also actually been done by other researchers[23, 9, 63]. Davis (1984) tries to cope with these kinds of faults in the domain of TTL-circuits by examining possible bridge faults among pins in the TTL-circuits that can cause the observed symptom. This technique is useful in the domain of digital circuits, but is difficult to be applied in other domains. Preist & Welham (1990) propose a similar solution as Davis's approach but based on the explicit representation of all potential unintended interaction paths. Due to the increase of model complexity, this approach has a very limited scope of applicability. Bötcher (1995) applies the *Hidden Interaction Detection* technique using *hidden interaction models* to represent structural faults. In principle, the Bötcher's approach will generate hidden interaction models among

components when it should generate structural faults. Hidden interaction models themselves are constructed by considering the *constellation description* among components and the *mode-constellation* of them. Here, the constellation description can be regarded as the topological relative position in my framework, while the mode-constellation is the behavioral modes of the components. The approach is different from mine in that Bötcher's approach will detect possible structural faults after it can detect the behavioral modes of all components while my approach will detect structural faults during generating fault hypotheses. In some domains, it is somewhat difficult to recognize the behavioral modes of all components in the device.

From the exploited mechanism of generating fault hypotheses, it can be understood that the fundamental work of the hypothesis generator is, given a symptom, analyze the symptom based on its type and by exploiting the information available in the provided strategies and sorts of knowledge corresponding to the type, generate fault hypotheses. This turns out to be a compact and fast mechanism to generate fault hypotheses; but, there is a shortcoming, that is, the mechanism will lead to producing inconsistent fault hypotheses. To some degree, we can improve the mechanism in order to avoid this problem, but the problem itself is still there and sometime the efficiency of the generator comes to decrease. Therefore, rather than improving the mechanism, it would be better if we can prune the inconsistent fault hypotheses. In the next chapter, I show how to prune inconsistent fault hypotheses.

Chapter 5

Hypothesis Testing

5.1 Introduction

So far I have outlined the framework for the first subtask, hypothesis generation, at which crucial sorts of knowledge and strategies for diagnosis are covered. Although by generating fault hypotheses all possible faults can probably be obtained, it wastes time if those all generated fault hypotheses, among which there are probably some inconsistent ones, have to be considered. Fault hypotheses must be considered as few as possible. This means that generated fault hypotheses should be tested before they are classified.

In this chapter, in order to provide a framework for the second subtask, hypothesis testing, I discuss what useful notions should be considered to test fault hypotheses generated by the hypothesis generator. I suggest an approach to pruning inconsistent fault hypotheses from the generated ones based on three types of fault hypotheses: pseudo fault hypotheses, contradicting fault hypotheses, and candidate faults.

5.2 Overview

The proposed framework for hypothesis testing can be shown in Figure 5.1. In principle, the hypothesis tester tries to prune pseudo fault hypotheses and contradicting fault hypotheses by means of the content of the fault hypotheses themselves or by checking if there are contradictions among the generated fault hypotheses, respectively. Furthermore, the hypothesis tester verifies also candidate faults constructed from fault models, by simulating the behavior of the device based on the fault models, device models, and the process models and checking to see if the simulated behavior can account for the observed symptoms or not.

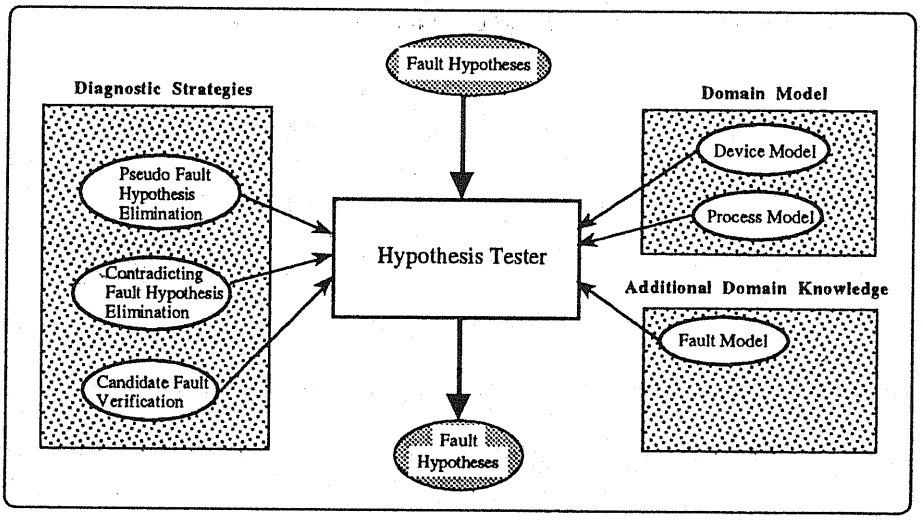


Figure 5.1: Hypothesis Testing

5.3 Pseudo Fault Hypotheses

As I have explained in the previous chapter, the fundamental work of the hypothesis generator is, given a symptom, analyze the symptom based on its type and by exploiting the available strategies and sorts of knowledge corresponding to the type,

generate fault hypotheses. Thus, for example, given *compression(liquid,compressor)* as a symptom, the hypothesis generator first selects *compression(X,Y)* from the process model then instantiates X and Y with liquid and compressor, respectively, obtaining *compression(liquid,compressor)* as shown in Figure 5.2.

compression(liquid,compressor)	
individuals:	*liquid, compressor*
initiating conditions:	*exist(liquid, compressor)*
sustaining conditions:	*addpress(compressor)* = +
effects:	*press(gas)* = +

Figure 5.2: Process compression(liquid,compressor)

It then makes use of information from the first three slots of the process to generate new symptoms or fault hypotheses. Finally, the following fault hypotheses will be generated:

f_hypothesis(liquid) (1)

f_hypothesis(compressor) (2)

f_hypothesis(exist(liquid,compressor)) (3)

f_hypothesis([addpress(compressor),+]) (4)

where *f_hypothesis* refers to a predicate indicating a fault hypothesis, and the argument of the fact created by the predicate denotes the content of the fault hypothesis.

From the above example, it can be understood that the hypothesis generator simply puts information from slots of a process to be fault hypotheses without further consideration. Consequently, some pieces of information from those slots that in fact do not suggest fault hypotheses will be automatically set to be fault hypotheses. This is what happened to *liquid* and *compressor* in the individuals slot above, which are merely a substance and a component, respectively, resulting in fault

hypothesis (1) and fault hypothesis (2). Fault hypotheses of this kind are what are called pseudo fault hypotheses. These pseudo fault hypotheses are necessary to be pruned because they do not give useful information for diagnosis. To prune them is not a difficult task. The hypothesis tester only needs to check to see if the content of a generated fault hypothesis is a substance or a component. If so, the hypothesis tester can simply get rid of it.

One may argue if it will not be better if the hypothesis generator is made not to generate fault hypotheses by means of information from the individuals slot. At a glance, this seems to be more efficient than checking each generated fault hypothesis and then discarding the pseudo one. However, the problem is that not always individuals of a process are substances or components. Some processes may have other processes as their individuals. Therefore, ignoring information from the individuals slot may cause to throw away significant fault hypotheses.

Given a process symptom, the hypothesis generator sometimes may also generate another type of pseudo fault hypothesis. Unlike the pseudo fault hypothesis (1) and (2) above, this pseudo one does not contain a substance or a component but indicates a normal state or action in a component. For example, consider the following context where the hypothesis generator is given *condensation(gas,compressor)* as a symptom. In a similar way as above, the hypothesis generator eventually obtains process condensation(gas,compressor) and generates fault hypotheses based on information in the first three slots of the process. Thus the generated fault hypotheses will be as follows:

f_hypothesis(gas) (5)

f_hypothesis(compressor) (6)

f_hypothesis(exist(gas,compressor)) (7)

f_hypothesis(flowing(heat,compressor,Dst)) (8)

Fault hypothesis (5) and fault hypothesis (6) can then be simply pruned because they are in the first type of pseudo fault hypothesis, and the possibilities of fault hypothesis (7) and fault hypothesis (8) to be candidate faults can then be considered. Fault hypothesis (7) suggests that *exist(gas,compressor)* be a fault hypothesis. However, from the device model of a compressor shown in Figure 5.3, it is weird if it must be regarded as a fault hypothesis. This is because there is

compressor	
purpose:	
state 1:	$outpress(gas) = +$
processes:	$compression(gas, compressor)$
terminal:	
input 1:	
connection:	*conduit* 4
parameters:	$inflow(gas), inpress(gas), intemp(gas)$
output 1:	
connection:	*conduit* 1
parameters:	$outflow(gas), outpress(gas), outtemp(gas)$
constraints:	
state 1:	$[inpress(gas)] + [addpress] = [outpress(gas)]$
	$[addpress] = POS\_CONST$
	$[inpress(gas)] = M_+[intemp(gas)]$
	$[outpress(gas)] = M_+[outtemp(gas)]$
	$[inflow(gas)] = M_+[outflow(gas)]$

Figure 5.3: A device model of a compressor

compression(gas,compressor) in the processes slot of the device model that has *exist(gas,compressor)* as an initiating condition, which means that there is gas existing in the compressor in the normal state. Besides being redundant information, this fault hypothesis can cause the hypothesis generator to generate more useless fault hypotheses based on it unless it is be pruned as quickly as possible. Thus, fault hypothesis (7) should be pruned as well, and in this case the only possible fault is fault hypothesis (8).

In order to prune a pseudo fault hypothesis like fault hypothesis (7), the hypothesis generator needs to check if the fault hypothesis has something to do with a process set in the processes slot of the device model of the corresponding component. If so, the hypothesis generator can then soon discard the fault hypothesis.

5.4 Contradicting Fault Hypotheses

Reasoning by exploiting different sorts of knowledge that are independent of each other can lead to produce some results that contradict each other. This cannot be avoided. What can be done is just find contradicting results, test which of them is consistent, and reject the inconsistent ones.

As explained in the previous chapter, the hypothesis generator exploits multifarious sorts of knowledge and strategies that are independent of each other to some degree for the purpose of generating fault hypotheses. Consequently, given some certain symptoms, it sometimes produces fault hypotheses among which there are two or more fault hypotheses contradicting each other. For instance, given *not_exist(gas,condenser)* as a symptom, by using naive physics as shown in Figure 5.4, the hypothesis generator first activates *flowing(gas,condenser,Dst)* and *condensation(gas,condenser)* in turn. It then sets them to be new symptoms and begins scanning each slot of both the two processes. From *condensation(gas,condenser)* symptom, whose slots similar to those of *condensation(gas,compressor)* in Figure 5.5, the hypothesis generator eventually finds *exist(gas,condenser)* in its initiating conditions slot and then simply makes it to be a fault hypothesis. In terms of temporal

$$not\_exist(gas, X) \Leftarrow flowing(gas, X.Dst) \bigvee condensation(gas, X)$$

Figure 5.4: Naive physics: not_exist(gas,X)

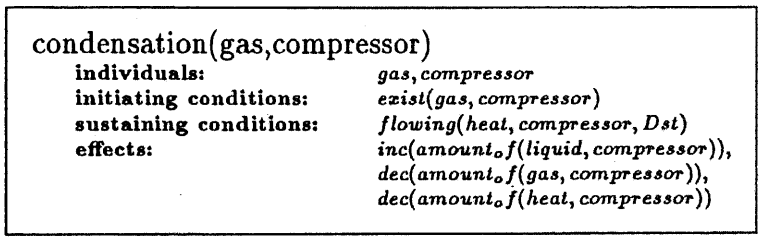


Figure 5.5: Process model of gas condensation in a compressor

logic, this fault hypothesis can be considered not to contradict the first given symptom, *not_exist(gas,condenser)*, since both fault hypotheses chronologically occur in different time. However, for the purpose of diagnosing faults, it can be regarded that both *not_exist(gas,condenser)* and *exist(gas,condenser)* are two fault hypotheses contradicting each other and the latter can then be gotten rid of in order to make the hypothesis generator not produce spurious fault hypotheses from it.

To realize this pruning method, the hypothesis generator needs to check whether or not the current fault hypothesis contradicts its ancestors in the currently generated fault tree by comparing it to each of its ancestors by turns. If it finds that the current fault hypothesis contradicts one of its ancestors, it can then immediately discard the current fault hypothesis.

5.5 Candidate Faults

Knowing how a component malfunctions has more positive impact on diagnosis than only knowing that a component is faulty. This will be more really felt, particularly, if the task of diagnosis is also aimed at helping take countermeasures to cope with faults. For instance, focusing on a pipe in a refrigeration plant if you find the input flow of the pipe normal but find the output flow of the pipe abnormal, say, decreasing, you only know that there is something wrong inside the pipe. Although this

is sufficient information for diagnosis, it is not enough information if you want to repair the fault. However, if you know that the pipe is either clogged or leaky, you can now repair it easily. For example, you can clean the inside of the pipe if you find the pipe clogged; or, you can mend the pipe if you find it leaky. From the above reason, I believe that a good diagnostic system should provide such information as well. To distinguish such information, i.e., information on how components malfunction, from fault hypotheses, which are usually physical states of components or processes in components, I term such pieces of information candidate faults. As can be understood from the example above, candidate faults can then be considered to be the specification of fault hypotheses. Accordingly, the first problem we face is then how to obtain candidate faults.

As can be seen from Figure 5.3 and Figure 5.5, there is no information on how a component malfunctions either in process models or in device models. This points out that we need another sort of knowledge in order to obtain candidate faults. For this purpose, I simply utilize what we have already known in the field of diagnosis as Fault Models. Figure 5.6 presents an example of fault models for a conduit in a refrigeration plant whose output flow is decreasing. It describes that a conduit that is clogged, leaky, or faulty can cause its output flow to decrease. In this fault model, *faulty(Conduit)* particularly serves as a dummy candidate fault representing all unknown fault models of the conduit. We need to provide such a candidate fault because it is important to make a fault model complete.

$$[Conduit, outflow(Subs), -] \Leftarrow clogged(Conduit) \bigvee leaky(Conduit) \bigvee faulty(Conduit)$$

Figure 5.6: Fault model: *[Conduit,outflow(Subs),-]*

Having had candidate faults of a fault hypothesis, we then need to verify each

candidate fault, that is to say that we need to test each candidate fault if it can account for the state caused by the symptom or not. This task is not so simple as the testing of the two types of fault hypotheses explained in previous subsections. We can not easily prune candidate faults only by considering their contents or checking among them if there are contradictions or not. To verify a candidate fault, at least we have to simulate the behavior of the device under consideration based on it. We then have to check if the behavior achieved from the simulation explains the state of affairs caused by the symptom. If so, we can promote the candidate fault; otherwise, we can discard it. Considering that fault models are difficult to be represented quantitatively, and even sometimes numerical values of parameters characterizing them are not precisely known, I exploit a diagnostic strategy applying qualitative simulation[25, 28, 37, 50, 51, 103] to do the simulation. Figure 5.7 depicts the framework I apply to verify candidate faults. As can be seen from the figure, in this frame work, I apply QSIM[51] and QPT[37] as the qualitative simulators.

The process of verifying candidate faults in the framework can be explained briefly as follows. First, given fault hypotheses generated by the hypothesis generator, the Candidate Fault Generator generates candidate faults by exploiting fault models. Since as its input, the QSIM/QPT-Based Simulator can take only qualitative differential equations (QDEs) and processes, it is necessary to interpret and model those candidate faults in terms of QDEs and processes. The QDE/Process Modeler does this task using the available process models and device models. Given the resulted QDEs and processes, the QSIM/QPT-Based Simulator then predicts the behavior of the device under consideration that corresponds to the QDEs and processes. Finally, the Behavior Matcher compares the behavior predicted by the QSIM/QPT-Based Simulator with the actually observed behavior of the device, checking if some values of interesting parameters in the predicted behavior qual-

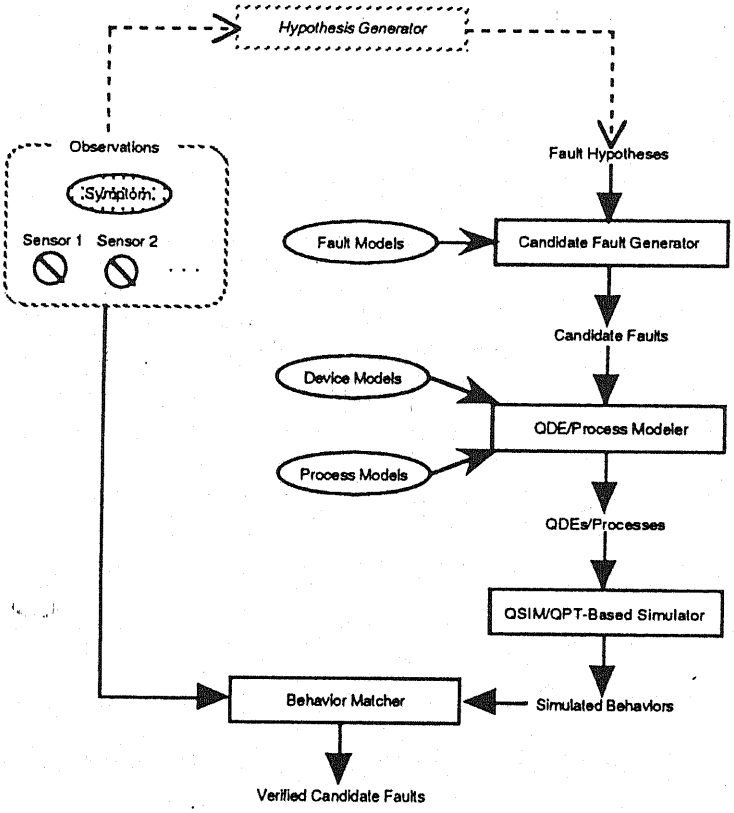


Figure 5.7: A framework for verifying candidate faults

itatively match the observed symptom and values of sensors in the device. If it finds some values of interesting parameters can account for the observed symptom and values of sensors in the device, the Behavior Matcher will promote the modeled candidate fault; otherwise, it will reject the candidate fault and discard it.

5.6 Discussion

One might argue that notions for testing fault hypotheses that I have proposed here only fit the hypothesis generator I proposed. Although in this paper I have focused on testing fault hypotheses generated by our hypothesis generator, notions involved in our approach to doing the task go further beyond it. If we recall the approach and reexamine it, we will see that the approach explains as well that in testing hypotheses, particularly fault hypotheses, which are generated by a generator, it is important to first exploit information available among those generated hypotheses themselves before requiring another information from outside. Particularly, this reduces the need for additional information from outside to do the task of testing, which results in gaining more efficiency in accomplishing the task. This is what the objective of classifying inconsistent fault hypotheses into pseudo fault hypotheses, contradicting fault hypotheses, and candidate faults is. I exploit information available among generated fault hypotheses by classifying pseudo fault hypotheses and contradicting fault hypotheses in order to reduce additional information we need in order to test candidate faults.

An objection to accepting that discarding pseudo and contradicting fault hypotheses is a task of a hypothesis tester might rise as well. I can accept this objection to some degree. Classical AI suggests that besides complete and nonredundant, a good hypothesis generator be informed[104]. Thus, owing to this, if our hypothesis generator is good it has to recognize such information on pseudo and contradicting

fault hypotheses beforehand and then do the task of discarding them by itself, instead of passing it to a hypothesis tester. If the task is done every time after the hypothesis generator generates one fault hypothesis, the task can be clearly regarded as a task of the hypothesis generator. In case of pseudo fault hypotheses containing substances or components, however, the task can not be done in such a way because it reduces efficiency in performing the task. Naturally, the task should be done after the hypothesis generator produces a certain number of fault hypotheses. Thus, in this case, the task can be hardly considered to be a task of the hypothesis generator but rather a task of a hypothesis tester. On the other hand, in case of pseudo fault hypotheses indicating normal states and of contradicting fault hypotheses, the task should be done directly after the hypothesis generator creates each of them because this prevents the hypothesis generator from generating spurious fault hypotheses from it. Therefore, in this case, the task should be done by the hypothesis generator. On the basis of these reasons, I have implemented the tester for pseudo fault hypotheses merely containing substances or components in a separate module, but I have intertwined the tester for pseudo fault hypotheses indicating normal states and for contradicting faults with the hypothesis generator.

Other work on testing generated fault hypotheses is involved in, for example, Pan (1984a, 1984b) and Davis (1984). Approaches to testing fault hypotheses involved in the two pieces of work differ markedly from mine in that they directly focus on testing candidate faults without trying classifying what kinds of fault hypotheses can be considered to be inconsistent. Pan suggests the predictive analysis methodology with state-transition models to verify candidate faults[59, 60]. The predictive analysis methodology is like the qualitative simulator in our framework for verifying candidate faults. However, it does not model QDEs or processes corresponding to candidate faults but describes the space of possible behaviors of components, under

both normal conditions and faulty conditions, in terms of a set of states, each of which explains a distinctive region of components' behaviors. In verifying a candidate fault, the methodology predicts state- transitions of a component indicated by the candidate fault and then matches them with signals from sensors through the qualitative feature abstraction technique. Instead of fault models, Davis applies constraint suspension in testing whether a component suspected to be faulty is consistent with all the observed behaviors of the device under consideration[23]. Although constraint suspension is useful in domains such as digital logic circuits, it provides little diagnostic power in a system consisting of complex constraints such as thermodynamic devices. Furthermore, it only explains that a component might be faulty but not how a component malfunctions. Davis argues that in performing the task of testing fault hypotheses, if we can not provide a complete list of fault models we will easily fall in the risk of discarding the real fault. To cope with this, I provide faulty(Component) serving as a dummy candidate fault representing all unknown fault models of the component. Thus, we do not need to be worried about getting rid of the real fault.

To sum up, in this chapter I have discussed significant aspects for testing fault hypotheses, classifying what kinds of inconsistent fault hypotheses should be discarded. I have identified pseudo and contradicting fault hypotheses that can be directly pruned and candidate faults that need verifying with qualitative simulations to check if they are consistent or inconsistent before they can be pruned. I have also indicated that candidate faults are important information that can help us take countermeasures to cope with faults, and pointed out that in order to obtain them, we need a sort of knowledge that is called fault models.

Chapter 6

Hypothesis Classification

6.1 Introduction

After we have all plausible fault hypotheses, what we still usually require is information that can guide us to make measurements on those fault hypotheses in order to find out the real faults quickly. Rather than Hypothesis Discrimination, I call this task *Hypothesis Classification* since I believe that what we should do in this final task is classify the plausible fault hypotheses into different ranked orders based on different criteria as needed.

In this chapter, in order to perform the task of hypothesis classification, I consider the use of the device model as the information of device structure, component observability, durability, and failure rates as criteria to rank fault hypotheses into some orders, and a binary split strategy to minimize the overall costs for probing.

6.2 Overview

A framework for hypothesis classification can be shown in Figure 6.1. To classify fault hypotheses, the framework considers Device Model as the domain model; and, Component Observability, Durability, and Failure Rates as the additional domain knowledge. The framework exploits a binary split strategy as a guide probe tech-

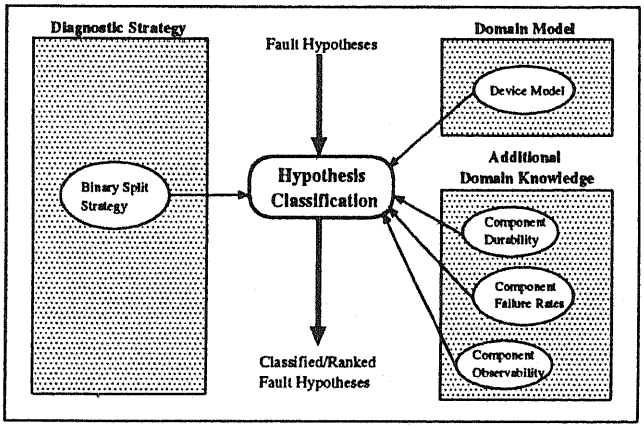


Figure 6.1: Hypothesis classification

nique in order to minimize the costs for probing.

In brief, given fault hypotheses, the hypothesis classifier classifies the fault hypotheses into expected orders based on component observability, durability, and failure rates. It then selects optimal test-points by exploiting binary split strategy and information on the device model.

6.3 Component Observability, Durability, and Failure Rates

In order to classify fault hypotheses into some expected orders, additional information is necessary. We can not do the classification only by means of the domain model of the device since it does not give us any useful information for classification. In diagnosis, component observability, durability, and failure rates are considered to be such kinds of information. There are, of course, other pieces of information that can be employed as criteria for classification; however, since the aforementioned sorts of information are usually available in the manual or the text book of the device, it is a proper step if we use them as the criteria.

The component observability of a component in a device can be measured with the degree of the observability of the component in the device. The easier we can observe a component in the device, the higher the degree of the observability of the component is. Although the component observability of each component in the device is not often explicitly provided in the manual or the text book of the device, we can specify it by examining the topology of the device and using a rule thumb, although not absolutely right, that is the deeper a component is located in the device the lower the component observability the component has. Thus, for example, in a portable TV set, the antenna has component observability higher than a transistor located in the amplifier system inside the TV set.

A component in a device can be said to be durable if the component is not easy to be the fault causing the device to fail to work normally. In other words, the component is not easy to be faulty. To indicate how durable a component in a device is, the notion of component durability is used. Unlike component observability, the component durability of a component is hard to be derived manually from the domain model of the device. Although sometimes we can specify it from the stuff composing the component, its availability really depends on given information. Consequently, if we cannot find information on component durability in the manual or text book of the device, and the information itself is not given, we cannot classify fault hypotheses based on it.

Another information that its availability depends on given information is failure rates. Failure rates indicate the failure probabilities of components in case the device does not work normally. They cannot be derived from the domain model either, but achieved from past experience. Sometimes, with the deep understanding of structure, behavior, and function of the device, we can derive the failure rate of each component in the device; however, the process itself is actually complicated

and is still somewhat far to be able to be automated with the current model-based reasoning technique. Since failure rates directly specify failure probabilities of components, in general situations they can be regarded as one of quick guided probe techniques.

Above are the three criteria that I use to classify fault hypotheses. On the basis of knowledge classification I have explained in the previous chapters, although the device partially and indirectly provides them but since they are used only for diagnosis, the three criteria can be classified into the additional domain knowledge class.

6.4 Classification Based on a Single and Multiple Criteria

Before we can classify fault hypotheses according to the provided criteria, we should first define how we represent the criteria. I represent each criterion with real numbers ranging from 1 to 10 with 1 as the lowest value and 10 as the highest value. Thus, for example, in case of component durability, component A with durability equal to 1 can be interpreted that the degree of durability of A is the lowest, while component B with durability equal to 10 can be understood that B has the highest degree of durability.

In case of classifying fault hypotheses based on a single criterion, the task is straightforward, that is to say, we can directly sort values of criteria available in the fault hypotheses. To give an example, assume that we are given a set of fault hypotheses: [fh1, fh2, fh3] where fh1, fh2, and fh3 account for failures on component 1, 2, and 3 in the device under consideration, respectively. Furthermore, given that component durabilities of component 1, 2, and 3 equals 5, 3, and 7, respectively, we can then classify the given set of fault hypotheses into ranked order set [fh2, fh1,

fh3], which indicates that in the process of probing we should first probe component 2, then component 1, and finally component 3.

How to classify fault hypotheses with more than two criteria? In order to classify fault hypotheses with multiple criteria, in principle we have add all values of criteria in each fault hypothesis in such a way that we can assign a new single value to each fault hypothesis. To do this task, I adopt a method of computing mean value from the probabilistic theory, by providing the following equation:

$$Cost(fh_i) = 1/n \sum_{y=1}^{n} F_{C_y}(Val(C_y))$$

where

- $Val(C_y)$ is the value of criterion y in the corresponding fault hypothesis,
- $F_{C_y}(Val(C_y))$ is the function to convert $Val(C_y)$ to a new value so that it can be added to values of other criteria,
- $Cost(fh_i)$ is the cost needed to probe fh_i where $i = 1, 2, 3, \cdots$.

As can be seen from the equation, we need a function that can convert the value of a criterion before we can add it to the value of another criterion. This is because with respect to the cost of probing, the order of numbers indicating the value of one criterion is not the same as that indicating the value of another criterion. For instance, value 1 of the component durability denotes that the degree of durability of the corresponding component is very low so that the probing of it should be done in the first priority. On the contrary, value 1 of the component observability means that the degree of observability of the component is very low indicating that the probing of it should not be done in the first priority. Therefore, it is obvious that we cannot add both the two value as they are, in order to obtain the average cost

of probing. In case of the chosen criteria: component durability, observability, and failure rates, the convert functions can then be simply represented as follows:

- $F_{C_{dur}}(Val(C_{dur})) = Val(C_{dur})$
- $F_{C_{obs}}(Val(C_{obs})) = 11 - Val(C_{obs})$
- $F_{C_{fail}}(Val(C_{fail})) = 11 - Val(C_{fail})$

where C_{dur}, C_{obs}, and C_{fail} are component durability, observability, and failure rates, respectively.

6.5 Binary Split Strategy

Although we already have ranked ordered sets of fault hypotheses, we can still minimize the total cost of probing by exploiting the notion of Binary Split Strategy. This is particularly very useful if there are lots of fault hypotheses with the same value cost in the ranked order sets; moreover, if we are placed in the worst situation where we do not have any information on criteria.

To show how binary split strategy can be applied to minimize the probing cost, assume we are given a simple cascaded inverters with a situation as shown in Figure 6.2. Given that the costs of probing at node a, b, c, d, and e are the same, and

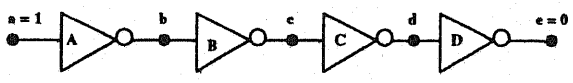


Figure 6.2: Cascaded Inverters

since the information flows unidirectionally from node a to node e, it is the best way of measurements if we half-split the information path, choosing node c to be the next probing node. This is because even in the worst case, we will only need 2

measurements to detect the faulty component. This way is better than the way of choosing node d or node b as the next probing node since the latter will result in 3 measurements in the worst case.

In the proposed hypothesis classification, during the classification process, after classifying fault hypotheses based on criteria, if the classifier finds fault hypotheses whose probing costs are the same or close it will try to examine if there are linear paths among those fault hypotheses by means of information on the device model. If it finds that there are linear paths among the fault hypotheses, it applies binary split strategy to reorder the fault hypotheses.

6.6 An Algorithm for Hypothesis Classification

An outline algorithm for hypothesis classification is then can be shown as follows:

```
INSERT a set of fault hypotheses

GET chosen ways of classifying fault hypotheses
IF      the chosen ways = single_criterion(Cx)
THEN
     DO      classifying fault hypotheses based on criterion Cy
     CREATE  a ranked order set of fault hypotheses
ELSE
IF      chosen ways = multiple_criteria([Cx,Cy,...])
THEN
     DO      calculating convert function for each criterion
        DO      classifying fault hypotheses based on multicriteria
     CREATE  a ranked order set of fault hypotheses
IF      there are fault hypotheses in the ranked order set whose
        probing costs are the same or close
THEN
             CALL  "Binary Split Strategy(the ranked order set)"
        ELSE
             CALL  "Binary Split Strategy(the set of fault hypotheses)"
```

As can be seen from the above algorithm, binary split strategy is also applied in case of no information on criteria available. Although the impact is not noticeable but at least, it will minimize the overall cost of probing.

6.7 Conclusions

This chapter has described a framework for hypothesis classification, exploiting component observability, durability, and failure rates as criteria for classifying fault hypotheses. It has indicated that a simple strategy, binary split strategy, can be used as a guided probe technique to minimize the overall cost for measurements.

Observability of a component can be calculated by means of topological relative position or by the degree of the observability of the component if looked at from outside the device, while the component durability and failure rates can be obtained from the text book or the manual of the device and from past experience, respectively. Although sometimes it is not so obvious, component observability usually can always be obtained since the device structure provides this information in principle. However, the conditions are not the same for component durability and failure rates whose availability really depends on text books and past experience. In case component durability and failure rates are not available, the framework, of course, cannot classify fault hypotheses based on them. But, this does not mean that it cannot perform the hypothesis classification. The framework can still do the hypothesis classification based on component observability and minimize the cost for probing by means of binary split strategy.

The next chapter discusses the implementation of the so far explained model-based diagnostic approach in a computer program called *MODEST* and shows how MODEST can detect structural faults as well as behavioral faults.

Chapter 7

MODEST: A Prototype MBD System

7.1 Introduction

To choose a domain to which the proposed framework is applied is not an easy matter. We have to carefully select a proper domain in order to know whether or not the proposed framework can be applied to real-world domains. Moreover, after we have got the domain, to determine if the application of the framework can be applied to solving real-world problems, choosing a suitable problem as an example can not be ignored either.

This chapter describes the implementation of the proposed framework in a computer program called MODEST and shows how MODEST can diagnose some possible faults in an extended elementary refrigeration plant—a simple but real domain. However, before coming to the explanations of MODEST, this chapter briefly clarifies an elementary refrigeration plant, a domain that was chosen to evaluate MODEST.

7.2 An Elementary Refrigeration Plant

The elementary refrigeration plant is the elementary mechanism of thermodynamic equipments such as refrigerators and air-conditioners. As shown in Figure 7.1, it consists of four main components: a compressor, a condenser, an expansion valve, and an evaporator, which are connected each other with a conduit, structuring a cyclic system. Besides those components, in the system there is also a refrigerant which, when the system is working, flows through the system in a thermodynamic cycle. To maintain the system, such as charging the system with the refrigerant when the system lacks it, each of the conduits and the components are equipped with one or more valves.

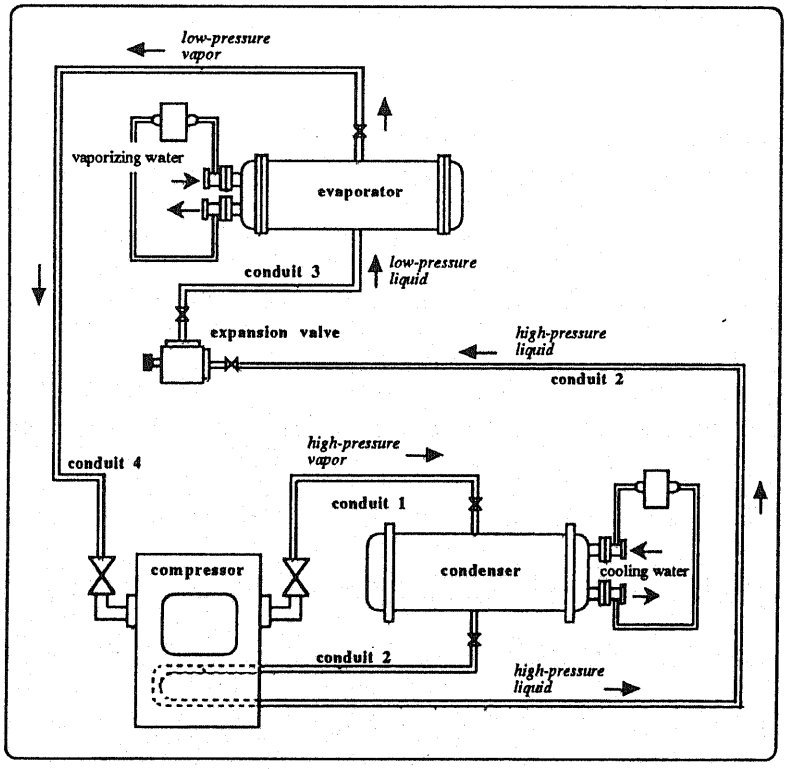


Figure 7.1: An elementary refrigeration plant

As can be found in[87, 13], the circulating system in the equipment can be

explained briefly as follows. The refrigerant enters the compressor as a slightly superheated vapor at a low pressure. It then leaves the compressor and enters the condenser as a vapor at a high pressure, where the refrigerant is condensed as a result of heat transfer to cooling water. The refrigerant then leaves the condenser as a high pressure liquid . Before going to the expansion valve, the liquid enters the compressor for a moment to cool the lubricating oil in the compressor. It then leaves the compressor and goes to the expansion valve. The pressure of the liquid is decreased as it flows through the expansion valve. It then enters the evaporator, and is vaporized as a result of heat transfer from vaporizing water. This vapor then enters the compressor.

Table 7.1 enumerates the size of sorts of knowledge used to model the chosen domain: an elementary refrigeration plant, and as additional information for diagnosing faults in the domain. As can be seen from the table, the size itself is relatively small compared with the real size of knowledge required to fully model

Table 7.1: The size of sorts of knowledge

Knowledge Sort		*Size* (pieces)
Domain Model	Device Model	8
	Process Model	13
	Topological Relative Position	3
Additional Domain Knowledge	Heuristics	5
	Naive Physics	6

actual thermodynamic equipments such as air-conditioners, refrigerators, etc. Nevertheless, since the sorts of knowledge used in that size represent the essence of that kind of device, the most important possible faults in the device can be captured as well in the model constructed by those sorts of knowledge. Currently, however,

we have been investigating other sources of knowledge on the device, such as what other components should be modeled, what other primitive processes, naive physics, and heuristics should be provided, etc., in order to gain higher performance of the implemented framework.

7.3 Implementation

In choosing a language for the implementation of MODEST, two main criteria are considered. Firstly, the language must be able to directly and easily code each sorts of knowledge and strategies in MODEST so that the rapid-prototyping of the system can be realized. Secondly, the language must be able to be used with ease to build the interface module, which is also not less important than the main system, as well. SICStus Prolog Ver. 2.1 #8 fulfills these conditions. MODEST has been written in SICStus Prolog running on a SPARC station. The basic structure of MODEST can be diagrammatically depicted as in Figure 7.2. As can be seen from the figure, like expert systems in general, MODEST has an interface module, an explanation module, an inference engine, a working memory, and a knowledge base. What is different from the usual expert systems is that MODEST has a diagnostic strategy module separated from the knowledge base. In the inference engine, MODEST has three main submodules, that is, the hypothesis generator, tester, and classifier, which are connected in order, while in the knowledge base, it has the domain model and additional domain knowledge. Table 7.2 shows the programs and their sizes.

Regardless of a somewhat long running time, we can get also other twofold benefits from implementing it in Prolog. The first is that we can almost directly code each sorts of knowledge and strategies in terms of *facts* and *rules* which are two basic statements in Prolog. The other is that we do not have to provide a list to maintain fault hypotheses in the course of generating fault hypotheses since we

Table 7.2: Component subsystems of MODEST

Component Subsystem	Description	Kilobytes of Source Files
Graphical User Interface	X-Window based GUI program	100
Explanation Module	Fault tree generation program	109
Hypothesis Generator	A program for generating fault hypotheses	80
Hypothesis Testing	A program for testing fault hypotheses	25
Hypothesis Classification	A program for classifying fault hypotheses	30
Domain Model	A program for coding the domain model	10
Additional Domain Knowledge	A program for coding the additional domain knowledge	10
Diagnostic Strategy	A program for coding the the diagnostic strategy	20

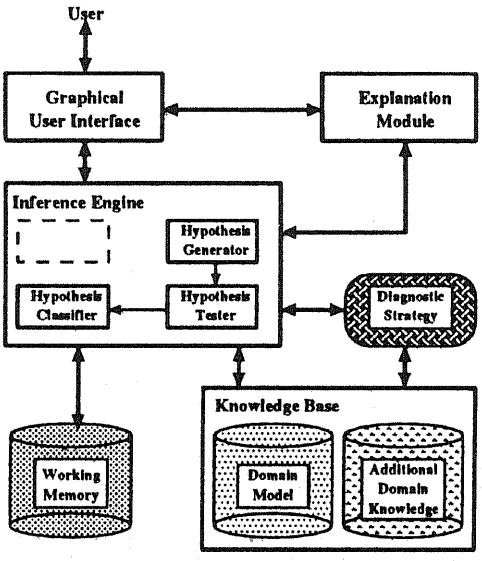


Figure 7.2: Overview of MODEST architecture

can insert them in the form of facts into the working memory used by the program.

Figure 7.3 shows an example of data structure of a device model of a compressor in terms of Prolog.

Other sorts of knowledge can be represented similarly, for example, as follows:

```
process(compression(gas,compressor),
  individuals([gas,compressor]),
  initiating_conditions([exist(gas,compressor)]),
  sustaining_conditions([[addpress(compressor),+]]),
  effects([[press(gas),+]])).

topological_relative_position(outdoor-unit,
  inside([compressor,propeller-fan,conduit-1,conduit-2,conduit-4])).

heuristics([knocking(compressor)],[compression(liquid,compressor)]).

naive_physics([exist(liquid,X)],[flowing(liquid,Src,X),melting(solid,X),
                                 condensation(gas,X)]).
```

Appendix C more completely presents some sorts of knowledge on the domain of refrigeration plants.

```
device(compressor,
  purpose([state-1(S_Cond1,[outpress(gas),+])]),
  processes([compression(gas,compressor)]),
  terminal([input-1(connection([conduit-4]),
            parameters([inflow(gas),inpress(gas),intemp(gas)]))],
           [output-1(connection([conduit-1]),
            parameters([outflow(gas),outpress(gas),outtemp(gas)]))]),
  constraints([state-1(S_Cond1,[add(inpress(gas),addpress,outpress(gas)),
            [addpress,+],
            m_plus(inpress(gas),intemp(gas)),
            m_plus(outpress(gas),outtemp(gas)),
            m_plus(inflow(gas),outflow(gas))])])).
```

Figure 7.3: A device model of a compressor in terms of Prolog

A similar way can be apllied to writing diagnostic strategies and the main program of the framework. However, a little problem usually appears in implementing a program in Prolog. In writing a program, we usually tend to use repeat-until routines rather than recursive routines in order to gain the efficiency of the program. Regrettably, there is no real repeat-until routines in Prolog.

Nevertheless, we can partly solve the problem. We can create a pseudo repeat-until routine in a prolog program by using two predefined functions: *repeat* and *fail*. Thus, in implementing the main program of the framework, we can divide it into two clauses, the first of which is to maintain the start point from which the main jobs have to be iterated and the other of which is to coordinate the main jobs themselves. For example, for hypothesis generation, they can be written as follows:

```
generating_fault_hypotheses(N):-
        repeat,
        get_symptom(Symptom,ID),
        gen_fault_hypotheses(N,Symptom,ID).
```

```
gen_fault_hypotheses(Symptom,ID):-
        (par_val_symp_analyzer(ID,Symptom)    -> true
                        ;
         heu_symp_analyzer(ID,Symptom)        -> true
                        ;
         pro_symp_analyzer(ID,Symptom)        -> true
                        ;
         naive_phy_symp_analyzer(ID,Symptom) -> true
                        ; true),!,
        ((ID is N - 1 ; not(active(Symptoms))) -> true
                        ; !, fail).
```

where N denotes the approximate number of fault hypotheses that should be achieved, and ID indicates the identity number of a symptom from which the fault hypothesis generation begins. The hypothesis testing and the hypothesis classification can be written in the similar manner as well.

A symptom or fault hypothesis is then can be represented in the following form:

```
f_hypothesis(ID,ParentID,Symptom).
```

where *ID* is the identity number of the symptom or fault hypothesis, *ParentID* is the identity number of the parent node of the symptom or fault hypothesis, and *Symptom* is the content of the symptom. Here, a single predicate of a symptom or a fault hypothesis, that is, *f_hypothesis*, is used in order to easily maintain fault hypotheses.

Therefore, once a user inserts a symptom into a working memory and sets the approximate number of fault hypotheses that should be achieved, the main program will be running and producing a list of fault hypotheses. That is, for example, when

a symptom that says knocking at the compressor *(knocking(compressor))* is entered, the following fault hypotheses will be generated:

```
f_hypothesis(0,*,knocking(compressor)).
f_hypothesis(1,0,compression(liquid,compressor)).
f_hypothesis(2,1,liquid).
f_hypothesis(3,1,compressor).
f_hypothesis(4,1,exist(liquid,compressor)).
.
.
.
```

To be more understandable, those then can be printed out in the form of *a fault tree*. The next section gives several examples of fault trees.

Often there are many difficulties in building an interface module with a language lacking graphics libraries. SICStus Prolog Ver. 2.1 #8 is equipped with the Graphics Manager, which is a package for graphics and interaction in the X-Windows environment, so that the main window of a system, menu bars, name labels, etc. can be created in an easy manner.

Appendix A and B give the prototype of the hypothesis generator and the applied diagnostic strategies, respectively.

7.4 Diagnosing A Structural Fault

Now, assume the part of conduit 2 inside the compressor leaks. It causes the refrigerant to leak out of the leak and to mix with the lubricating oil, which then makes the refrigerant be sucked into the compressor. This causes liquid compression at the compressor, which finally results in knocking at the compressor (See the broken line

in Figure 7.4). Owing to adding a new connection, i.e., a connection between the conduit 2 and the cylinder of the compressor, to the structure of the system, this can be regarded as a structural fault.

Here, we will show how MODEST diagnoses the fault. We have *knocking at the compressor* as a symptom. The sequence of the diagnosis of the structural fault can

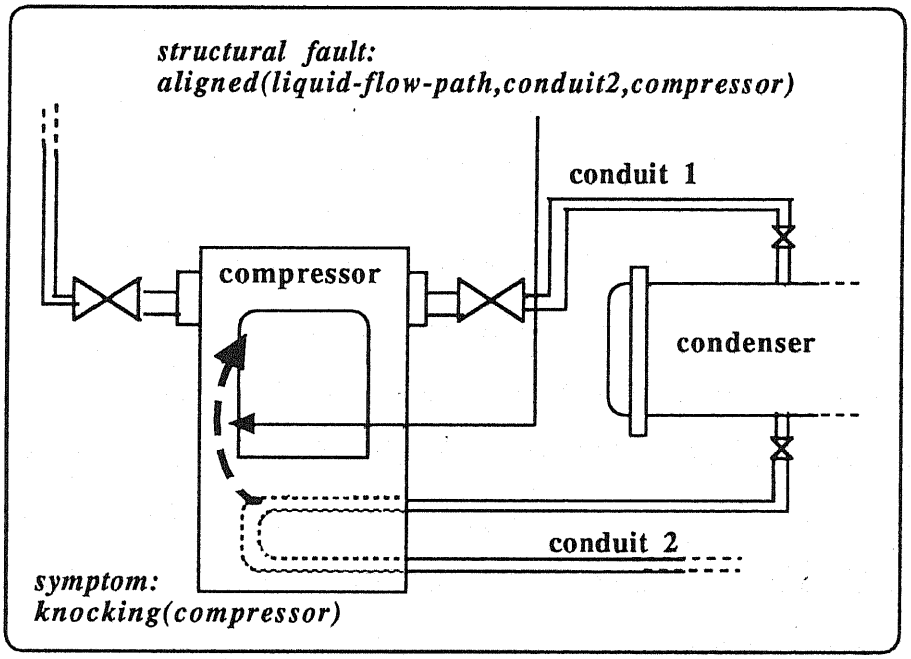


Figure 7.4: A structural fault inside a compressor

be explained as follows (Follow the diagnosis sequence shown in Figure 7.5, while reading the following explanation. The number in each step corresponds to the number written in the figure.):

1. First, the MODEST sees if the symptom is at the level of heuristics, the level of processes, the level of naive physics, or the level of parameter values. In this case, the symptom is at the level of heuristics. Using an abductive style,

the hypothesis generator then activates the left hand side of the corresponding rule, which leads to consider that process compression, i.e., liquid compression at the compressor, is active.

2. MODEST then sets the process as a new symptom and proceeds with its postdiction, scanning the slots of the process. In the initiating conditions, the hypothesis generator finds *exist(liquid,compressor)* and then makes it to be a new symptom.

3. This will move the focus of attention of MODEST to the naive physics, which leads to consider that process *flowing(liquid,Src,compressor)* was active (for simplicity, the other processes are ignored). In the same way as in the previous steps, the hypothesis generator then sets the process to be a new symptom.

4. MODEST then begins scanning the slots of the process. From the individuals, initiating conditions and sustaining conditions, it can be considered that there is a *contained-liquid* as the Src(i.e., something that keeps liquid, e.g.,tank, pipe, etc.) whose pressure is greater than the pressure of the compressor and there is a *liquid-flow-path* between the contained-liquid and compressor.

5. MODEST then searches which component in the device is possible to be the Src or the contained-liquid. This is done by searching the topological relative position or the space in which the compressor is. Finding the outdoor-unit as the space the hypothesis generator then checks which of the components: propeller-fan, conduit 1, conduit 2 and conduit 4, is possible to be the Src. By examining the corresponding device model, the hypothesis generator knows that there is a liquid in conduit 2 and also that only conduit 2 is not directly

connected to the compressor. This leads MODEST to deduce that there is *a new connection as a liquid-flow-path between conduit 2 and the compressor*, which leads us to think that perhaps *conduit 2 is leaky, and the liquid leaking out of conduit 2 flows into the cylinder of the compressor.*

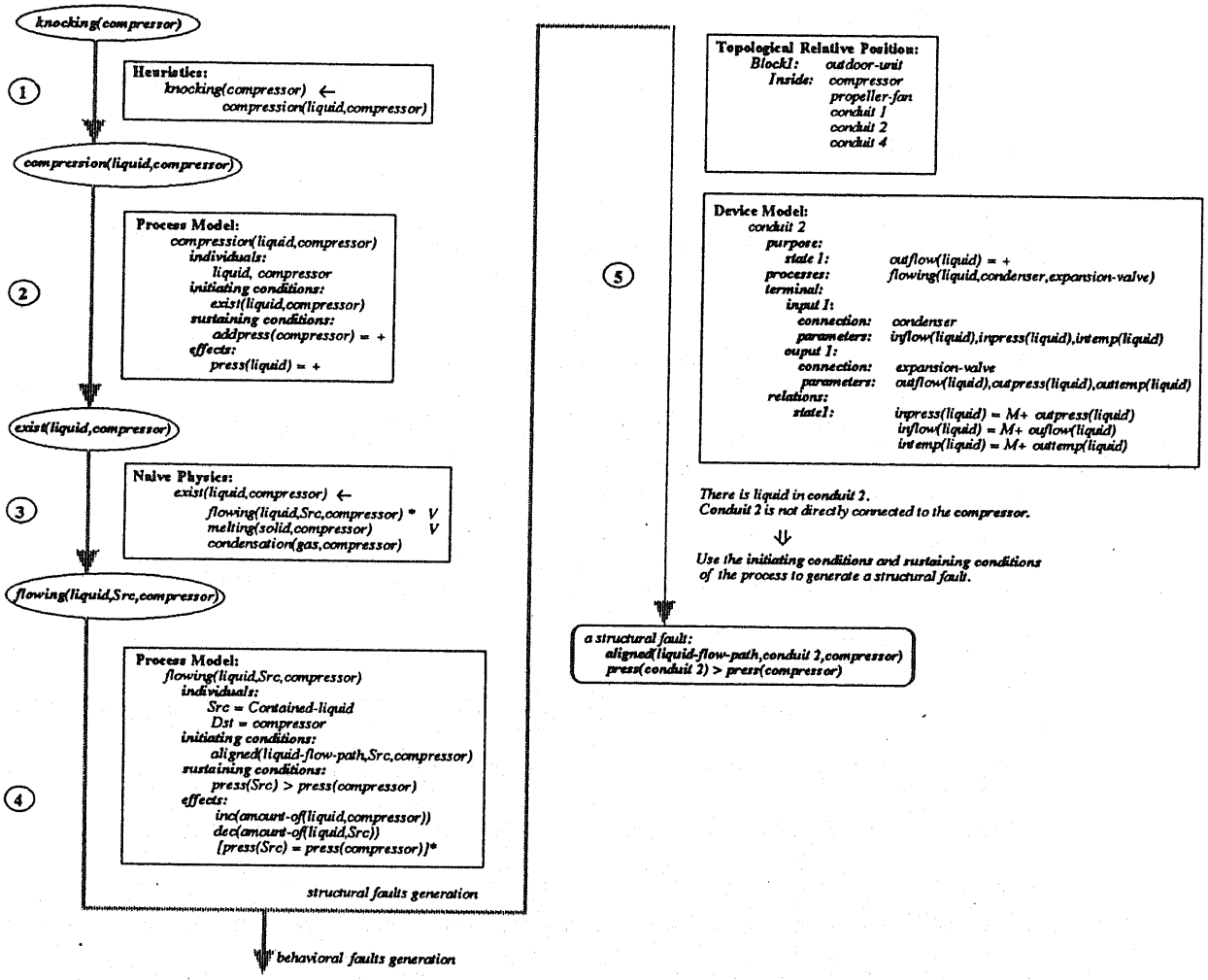


Figure 7.5: A diagnosis sequence

Actually, as can be seen from the hard copy of the output shown in Figure 7.6, the hypothesis generator generates more than five fault hypotheses, given the symptom. However, concentrating on detecting possible structural faults, we just focus on fault hypotheses with ID numbers: 1, 4, 6, 9, 10, 11, and 12.

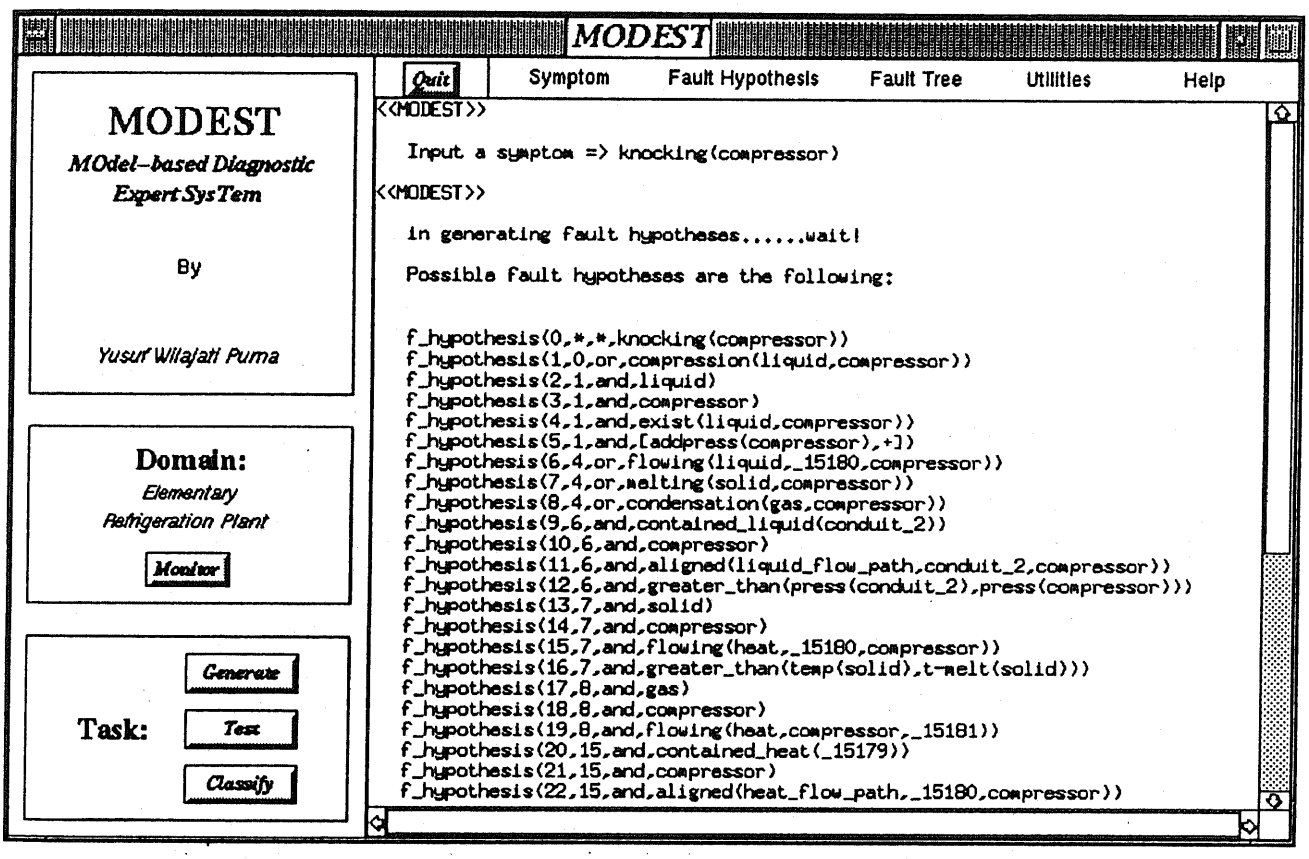


Figure 7.6: Fault hypotheses from symptom *knocking(compressor)*

Figure 7.7 shows a snapshot of a part of the fault tree produced from the output fault hypotheses.[1] Solid lines in the fault tree depict the path of a diagnosis sequence corresponding to that shown in Figure 7.5.

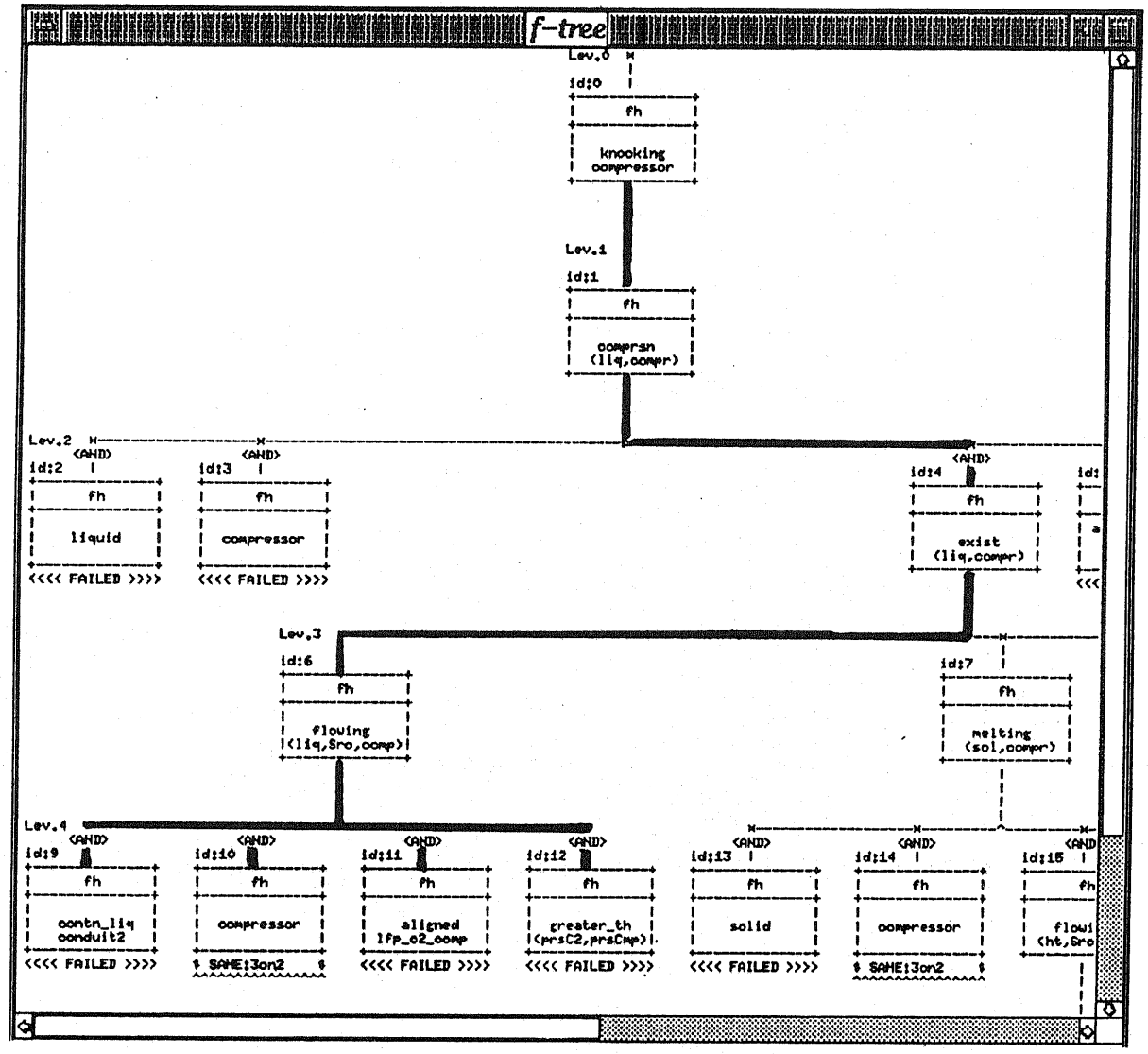


Figure 7.7: Part of the fault tree with symptom *knocking(compressor)*

[1]The label FAILED in a node explains that the node cannot be further explained, while label SAME:XonY in a node indicates that the content of the node is the same as that of the node with id:X on level:Y.

7.5 Diagnosing A Behavioral Fault

As can be found in [58], one of possible abnormal symptoms in a compressor is lubricating oil with a temperature higher than normal, which is a result of a very high temperature of the output gas of the compressor. It is stated that that is probably caused by a misset or a malfunctioning expansion valve, i.e., an overthrottled expansion valve (Figure 7.8).

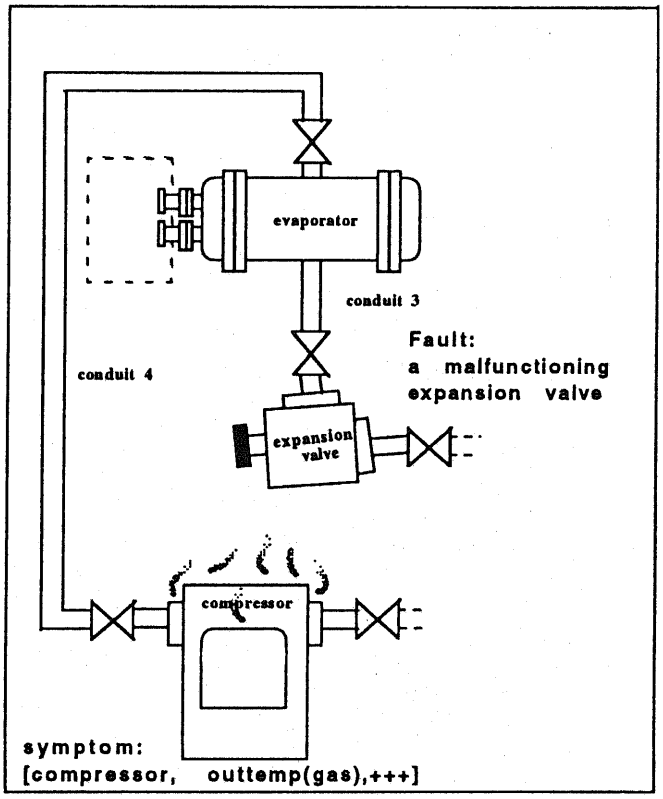


Figure 7.8: A behavioral fault in an expansion valve

In this section, I show that MODEST can deal with the aforementioned fault as well. Since we do not model the lubricating oil unit, we can interpret the symptom as the output gas of the compressor whose temperature is higher than normal. This can be represented in the following form:

$$[compressor, outtemp(gas), +++]$$

Given the symptom, MODEST generates arround 100 possible fault hypotheses, part of which are are shown by the hard copy in Figure 7.9. From the fault hypothesis with ID number 38, we can find that the symptom can be caused by the expansion valve that is overthrottled.

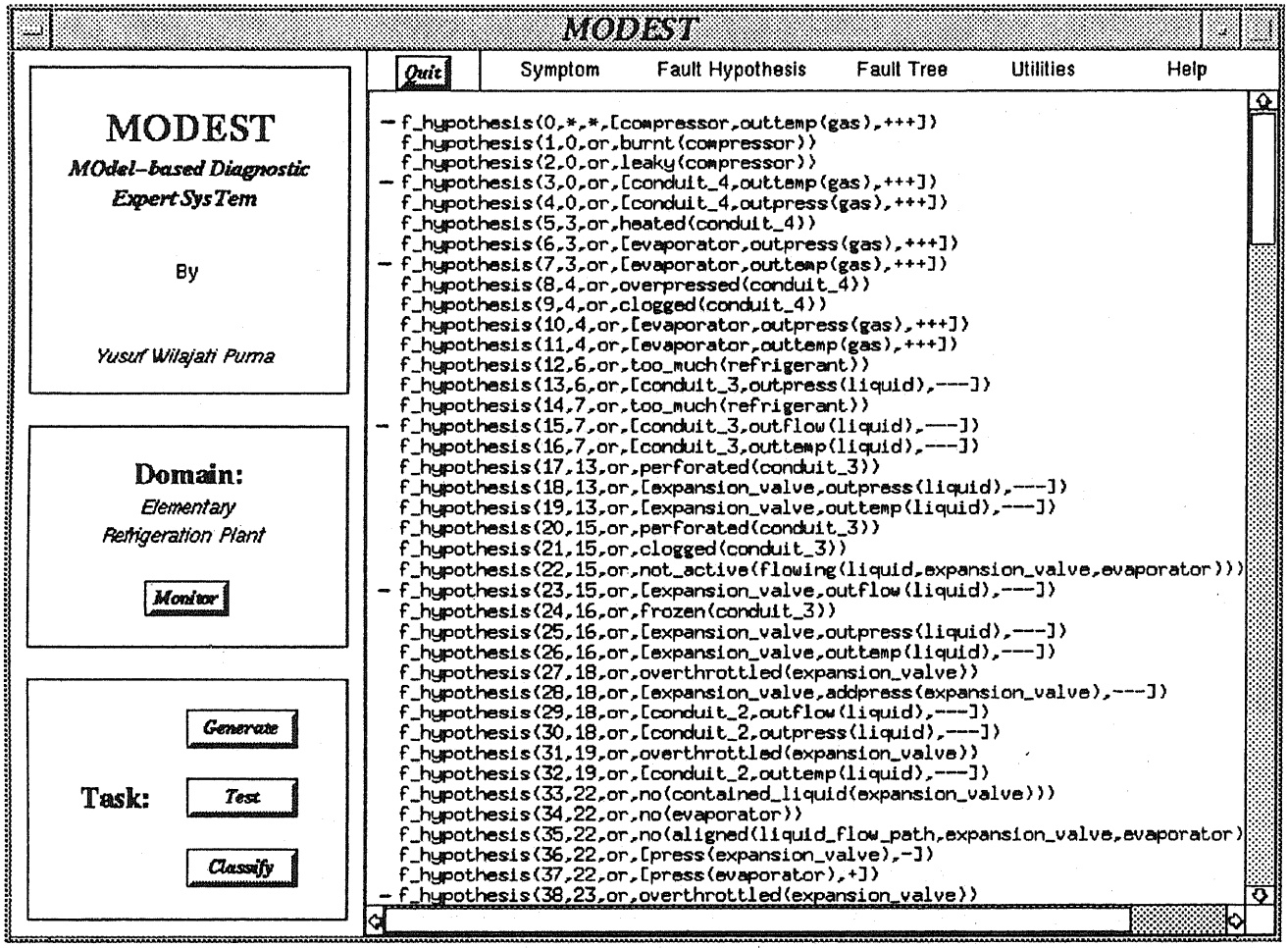


Figure 7.9: Part of Fault hypotheses from symptom *[compressor,outtemp(gas),+++]*

To be more understandable, it can be explained as follows. Let us trace back from the fault hypothesis with ID number 38. (Follow the path written in solid lines in each snapshot of part of the generated fault tree shown in Figure 7.10, 7.11, 7.12, and 7.13, respectively.) An overthrottled expansion valve, more precisely, an expansion valve that is closed more than usual, causes the amount of the liquid flowing

through the expansion valve to decrease (*id:23, [expansion_valve,outflow(liquid),- - -]*). This results in a very small amount liquid flowing out of conduit 3 which is connected to the expansion valve in the input and to the evaporator in the output (*id:15, [conduit_3,outflow(liquid),- - -]*). Since the liquid enters the evaporator in a smaller amount and receives heat almost the same as usual, the evaporator consequently ejects a gas with a temperature higher than normal to the conduit 4 (*id:7 [evaporator,outtemp(gas),+++]*). This indirectly results in the gas flowing out of conduit 4 with a high temperature (*id:3 [conduit_4,outtemp(gas),+++]*). Thus, the gas enters the compressor with a high temperature, which causes the temperature of the gas in the compressor to be high, which results in a high temperature gas in the output of the compressor (*[compressor,outtemp(gas),+++]*).

7.6 Testing Faults

I have tested the methods of pruning pseudo fault hypotheses and discarding contradicting fault hypotheses in MODEST by applying them to testing fault hypotheses that are generated by our hypothesis generator given several possible symptoms in the chosen domain. In this test, besides knocking(compressor) as an example of heuristic symptoms, and [compressor,outtemp(gas),+++], I have chosen [compressor,outpress(gas),- - -], and [compressor,inpress(gas),+++] as examples of parameter value symptoms. After generating fault hypotheses, given those four symptoms, and applying the two aforementioned methods, I got the results as shown in Table 7.3.

We can see from the table that discarding pseudo fault hypotheses from the fault hypotheses generated from symptom knocking(compressor) reduces the number of those fault hypotheses from 48 to 19. The reduction (almost 60% of the earlier number) is considerably greater than those of fault hypotheses generated from the other

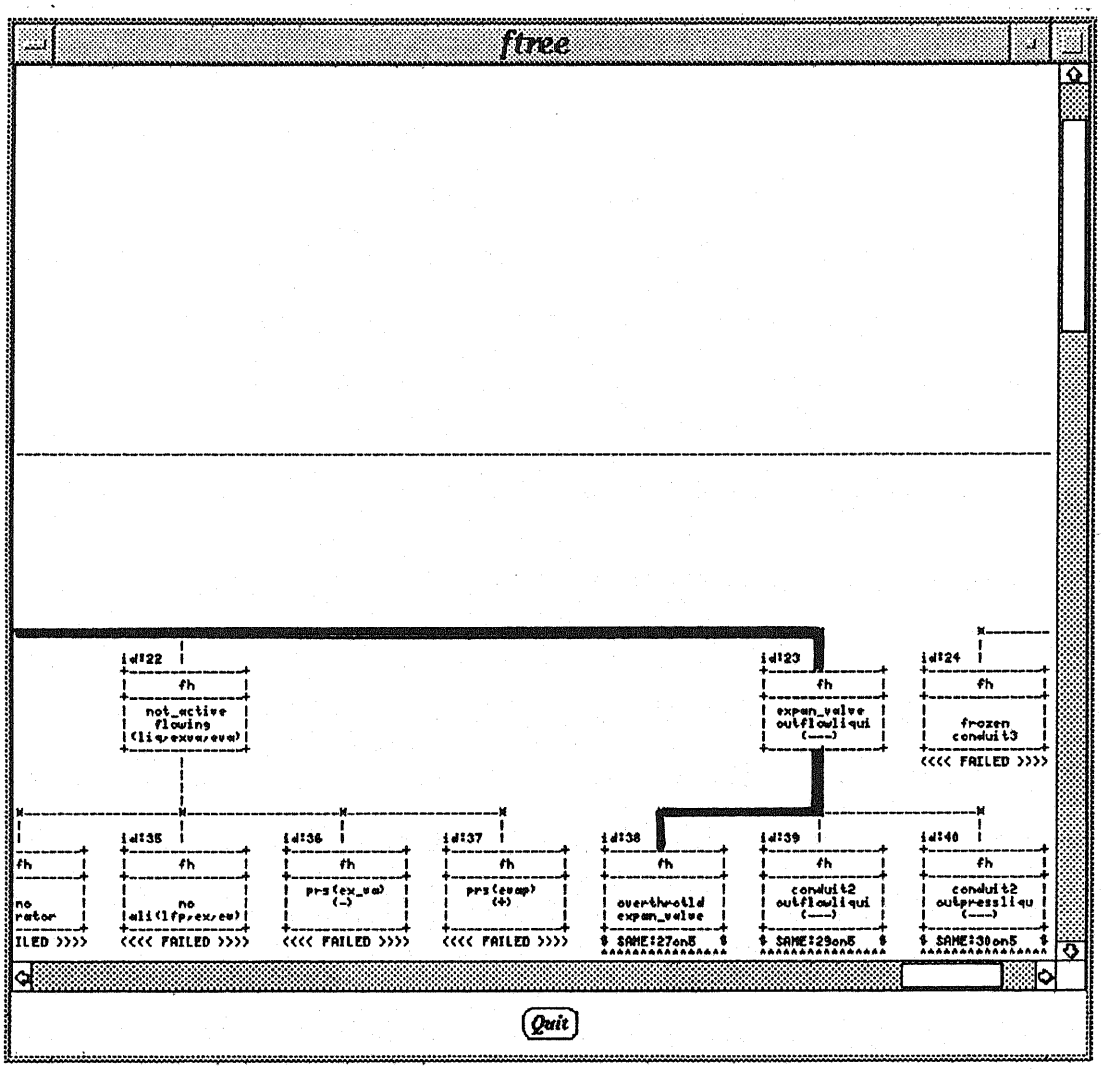


Figure 7.10: Part of a fault tree with symptom *[compressor,outtemp(gas),+++]* (A)

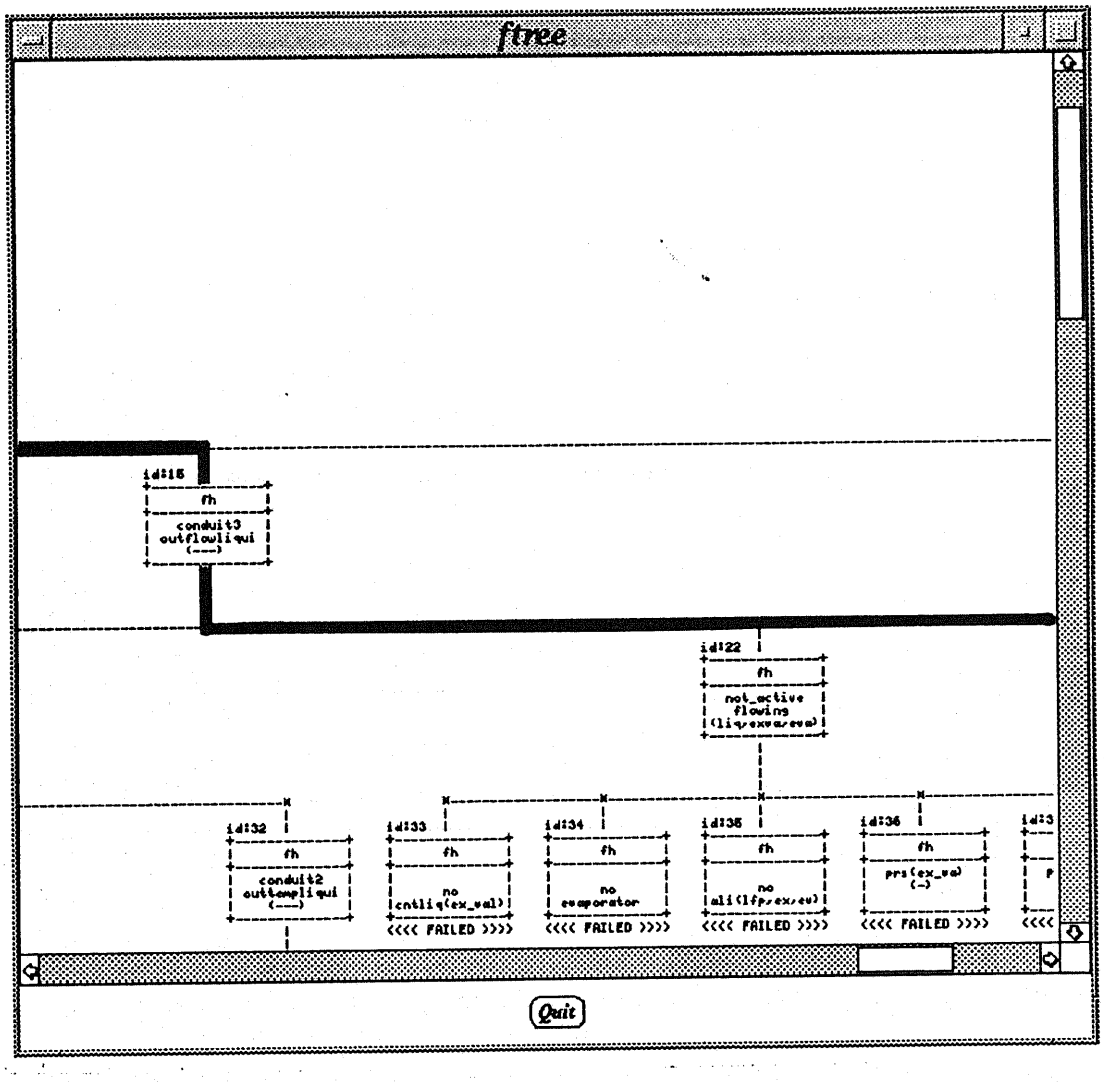


Figure 7.11: Part of a fault tree with symptom *[compressor,outtemp(gas),+++]* (B)

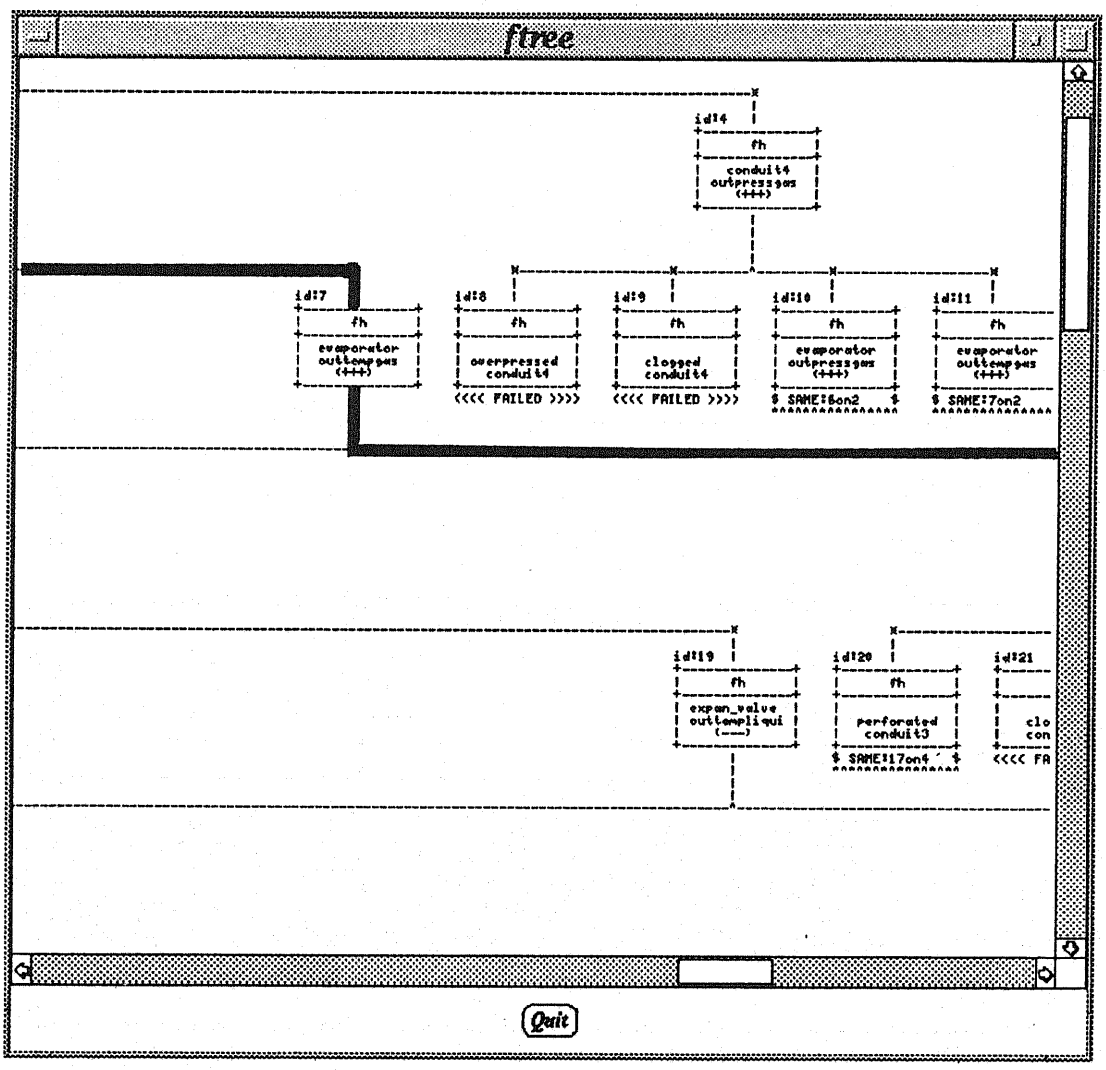


Figure 7.12: Part of a fault tree with symptom *[compressor,outtemp(gas),+++]* (C)

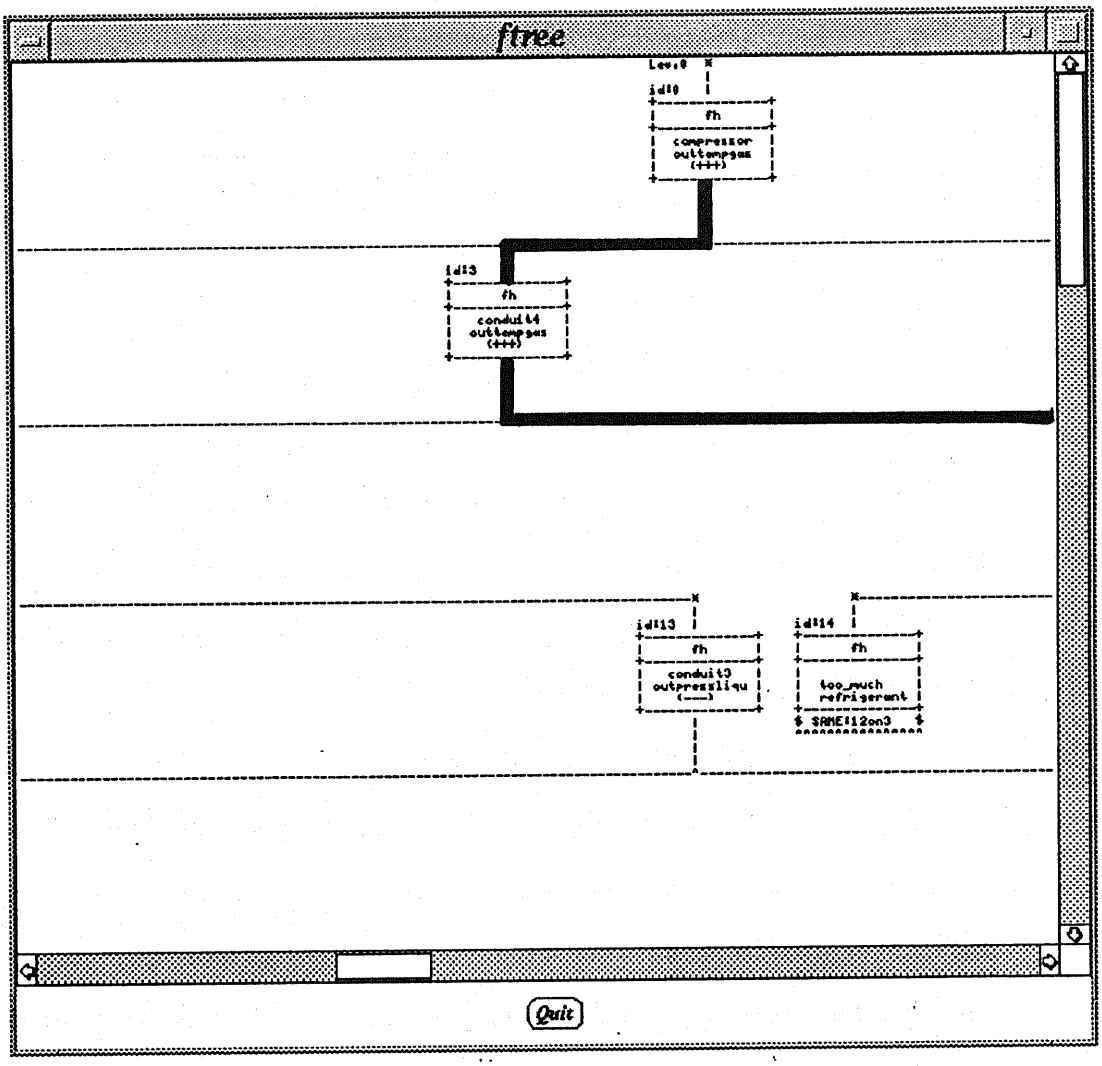


Figure 7.13: Part of a fault tree with symptom *[compressor,outtemp(gas),+++]* (D)

Table 7.3: Results of testing generated fault hypotheses

	Generated *fh*	*fh* after Pseudo *fh* Discarded	*fh* after Contradicting *fh* Discarded	*fh* after Pseudo & Contradicting *fh* Discarded
knocking(compressor)	43	19	43	19
[compressor,outtemp(gas),+++]	106	93	75	71
[compressor,outpress(gas),---]	125	114	94	92
[compressor,inpress(gas),+++]	107	94	76	72

three symptoms that are approximately only 10% of the earlier number. Although we must also consider the possibility of normal state fault hypotheses' being generated, the considerable reduction indicates that many processes have been probably activated by the hypothesis generator, when generating a diagnosis from the given symptom, which results in the generation of many fault hypotheses, each of which merely contains a substance or a component.

In contrast to the above, getting rid of contradicting fault hypotheses among the fault hypotheses generated from symptom knocking(compressor) does not reduce the number of those fault hypotheses at all. On the other hand, doing so to the fault hypotheses generated from the other three symptoms reduces more inconsistent fault hypotheses than discarding pseudo fault hypotheses from them. This shows that in generating fault hypotheses from those three symptoms, all of which are parameter value symptoms, the hypothesis generator probably has triggered naive physics and applied the qualitative value propagation many times, which results in the generation of contradicting fault hypotheses. Although from the results, we can not get the clear relationship between the type of a symptom and the number of fault hypotheses being reduced, since the number of fault hypotheses generally decreases

after discarding pseudo or/and contradicting fault hypotheses, we can consider that in the chosen domain the hypothesis generator generally produces either pseudo or contradicting fault hypotheses, when generating a diagnosis. Consequently, this shows that even for a simplified physical device, discarding pseudo and contradicting fault hypotheses can result in substantial gains in the performance of diagnosis, that is, we can obtain smaller numbers of fault hypotheses that must be considered.

7.7 Conclusion

This chapter has briefly described MODEST, the chosen domain: an elementary refrigeration plant and given two clear examples of diagnosing faults in the domain. The two examples have shown that the framework can partly cope with real-world problems, diagnosing a behavioral fault as well as a structural fault, with promising results.

Particularly, in diagnosing the structural fault, MODEST has shown that it can deal with a kind of fault which is difficult to tackle with other model-based diagnostic approaches.

Chapter 8

Conclusions

In this chapter, I summarize the key aspects of the research, describing the main features of the proposed framework and comparing them with other related proposals. Besides, I discuss limitations of the framework and present some directions for future work.

8.1 Summary

The main features of the framework can be summarized as follows:

- *The clear division of the task of diagnosis.* As a matter of fact, this feature should be addressed to the framework for the extended model-based paradigm as a whole rather than to the framework for hypothesis generation alone. This thesis has introduced three subtasks of diagnosis. Those are the hypothesis generation, hypothesis testing, and hypothesis classification. The advantage that can be taken of dividing the task of diagnosis into several simpler subtasks is that we can exclusively build a framework for each subtask, and then integrate all frameworks into a unit framework for diagnosis. This indeed reduces problems to be solved in building a diagnostic system. Moreover, the management of the diagnostic system based on this feature will be easier since

a different framework is for a different subtask.

Although several other approaches to diagnosis, or model-based diagnosis in particular, such as [23, 27, 105] pointed out steps that should be taken for diagnosing faults, they did not provide a framework for each step clearly. It is obvious, therefore, that the aforementioned benefit can not be taken from them.

- *The providing of clear classes of information useful for diagnosis.* It is natural that knowing classes of information required for diagnosis facilitates us to identify sorts of information of each class useful for diagnosis. Therefore, the clear classification of information is a requirement for getting the benefit. Besides, as the previous feature, this feature gives us facility in managing diagnostic systems. In this thesis, I have classified information for diagnosis into three classes: the domain model, additional domain knowledge, and diagnostic strategy.

  The research relating to classifying information useful for diagnosis has also been done by other researchers. For example, Abu-Hanna et al. classified knowledge in model-based diagnosis into two classes: the model knowledge class and the diagnostic class [2]. The two knowledge classes somewhat resemble the domain model and the additional domain knowledge of the framework this thesis proposed. However, the two knowledge classes differs from ones of the framework in sorts of knowledge that are included in each class. For example, a sort of knowledge such as naive physics is not available in the two knowledge classes.

- *The use of several sorts of knowledge and strategies for diagnosis.* This is a consequence of the preceding feature. What advantage can be be taken of

this feature is of course the enhancement of the diagnostic system ability to diagnose faults. The framework for hypothesis generation proposed by this thesis have identified five sorts of knowledge and three sorts of strategies for diagnosis. Those are the device model, process model, and topological relative position which are involved in the domain model; the heuristics and naive physics belonging to the additional domain knowledge; and the qualitative value propagation, direct path of causality, and structural fault localization as diagnostic strategies. As I have shown in this thesis, for example, incorporating sorts of knowledge such as the naive physics, process model, device model, etc. with the structural fault localization strategy can partly solve problems of diagnosing structural faults.

In contrast, pure model-based diagnostic approaches such as [23, 27] do not consider this feature. This causes difficulty to them in diagnosing structural faults.

- *The clear separation of knowledge from strategies for diagnosis.* From the point of view of problem-solving methods and sorts of knowledge for diagnosis, the diagnostic strategy can be separated exclusively from the domain model and additional domain knowledge, in the sense that the diagnostic strategy is related to problem-solving methods for diagnosis while the domain model and additional domain knowledge concern sorts of knowledge for diagnosis. Although I do not explain explicitly in this thesis, if we carefully look into the process of diagnosing, we will find that the success of the process also depends on how appropriately the diagnostic strategies and sorts of knowledge for diagnosis are compiled as needed. This means that the clear separation of knowledge from strategies is also a key for the success of a diagnosis.

Moreover, this feature can be considered to be a first step towards providing a framework for the compilation of strategies and knowledge paradigm. This can be hoped to be a good extension of the knowledge compilation paradigm proposed by [80].

- *The providing of various symptom representations.* Even though the framework does not put emphasis on the notion of symptom representations, it has classified symptoms into four types: the parameter value symptoms, process symptoms, heuristic symptoms, and naive physics symptoms. Although not directly, it has indicated that this classification can provide proper representations for describing different symptoms and gaining the efficiency of the framework itself.

  Generally speaking, this feature breaks the old tradition of model-based diagnostic approaches [23, 27, 105] using a single representation for representing symptoms.

However, nothing is complete. The framework still possesses a lot of limitations. This is the topic of the next section.

8.2 Limitations of the Framework

The most obvious limitations of the framework can be mentioned as follows:

- *The difficulty in representing complex devices.* Currently, in the device model, the framework uses a one-frame-to-one-component representational language. It is useful to represent a simple device. But, it will be impractical if the device to be represented is a complex device. This is because it is not enough to decompose the device into several components. In other words, it is necessary

to represent the device hierarchically. Hence, the current device model should be modified to be a hierarchically representational language.

- *The difficulty in generating multiple faults.* Since the framework just exploits the qualitative value propagation, direct path of causality, and structural fault localization as diagnostic strategies, it is natural that the framework finds generating multiple faults difficult. Therefore, a diagnostic method such as applied by GDE should be taken into consideration.

- *The difficulty in diagnosing intermittent faults.* In some domains, possible faults may be dynamic. Thus, a component works correctly for a while and exhibits a faulty behavior in other periods. It is quite difficult to cope with such a kind of fault since the symptom cannot be simply interpreted. It maybe sometimes higher than normal and sometimes lower. We need a method to interpret this kind of symptom and to propagate it in order to detect which components are faulty.

- *The difficulty in diagnosing complex feed-back loop mechanisms.* If we look further into the domain of refrigeration plants I have used here, it turns out that the domain is actually a loop system. In a big loop system, sometimes we can cut the system on some point so that we can regard it as a linear system. However, if in a system, there are many loop-back mechanisms it is not sufficient if we model the system as as a linear system. We need a different strategy to cope with loop-back mechanisms.

8.3 Future Work

Most of things left are developing a bigger prototype of the framework, evaluating the framework by testing it in diagnosing other possible faults, and evaluating the

framework for other domains. Evaluating the system with real human experts is of course very important, particularly to bring the system into the real application world. Furthermore, for the sake of convenience and portability, although it is not the main focus of the research, implementing a prototype in a personal computer level should be considered as well. Thus, the first future work will be concentrated on them. Besides, addressing the aforementioned limitations will be taken into account in order to enhance the performance of the framework.

Appendix A

Hypothesis Generator

This appendix presents the source program of the prototype of the hypothesis generator. As can be seen from its structure, it is wholly implemented in Prolog. Each symptom analyzer is headed by its name.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                              %%
%%    HYPOTHESIS GENERATOR      %%
%%                Ver. 1.0      %%
%%      wila, Feb 16, 1993      %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

start :-
        write('----------------------------------------'),nl,
        write('HYPOTHESIS GENERATOR  Ver. 1.0 -- Wila,93'),nl,
        write('----------------------------------------'),nl,
        write('Input a symptom => '),read(Symptom),nl,
        write('Input the approximate number of fault hypotheses => '),
        read(N),nl,
        abolish(id,1),
        abolish(active,1),
        abolish(cp,1),
        abolish(f_hypothesis,3),
        assert(id(0)),
        assert(active(0)),
        assert(f_hypothesis(0,*,Symptom)),
        write('in generating fault hypotheses ..... wait !'),nl,nl,
        generating_fault_hypotheses(N),
        write('Possible fault hypotheses are the following:'),nl,
        listing(f_hypothesis).

generating_fault_hypotheses(N) :-
```

```
        repeat,
        get_symptom(Symptom,ID),
        gen_fault_hypotheses(N,Symptom,ID).

gen_fault_hypotheses(N,Symptom,ID) :-
        (par_val_symp_analyzer(ID,Symptom)    -> true
                        ;
         heu_symp_analyzer(ID,Symptom)        -> true
                        ;
         pro_symp_analyzer(ID,Symptom)        -> true
                        ;
         naive_phy_symp_analyzer(ID,Symptom) -> true
                        ; true),!,
         ((ID is N - 1 ; not(active(Symptoms))) -> true
                ; !, fail).

get_symptom(Symptom,ID) :-
        f_hypothesis(ID,_,Symptom),
        active(ID),
        retract(active(ID)),!.

%----------------------------%
%  Parameter Value Analyzer  %
%----------------------------%

par_val_symp_analyzer(ID,[Dev,Var,Val]) :-
        is_devpurposepar(Dev,Var) -> sol_for_dev_purposepar([Dev,Var,Val],ID)
                                  ;
        is_devpar(Dev,Var) -> qvprop_dirpath([Dev,Var,Val],ID)
                                  ;
        true.

is_devpurposepar(Dev,Var) :-
        device(Dev,purpose(Purpose),_,_,_),!,
        Purpose = [state_1(S_Cond1,[Var,Val])].

sol_for_dev_purposepar([Dev,Var,Val],ID) :-
        device(Dev,_,processes(Plist),_,_),
        set_not_active(Plist,NotProc),
        insert(NotProc,ID).

set_not_active([],[]) :- !.
set_not_active([P|Ps],[not_active(P)|NPs]):-
        set_not_active(Ps,NPs).
```

```
is_devpar(Dev,Var) :-
        device(Dev,_,_,_,constraints(Const)),
        Const = [state_1(S_Cond,Constraints)],
        member_cons(Var,Constraints).

member_cons(_,[]) :- !.
member_cons(Var,[C|Cs]) :-
        ((C = m_plus(X,Y) ; C = m_minus(X,Y) ;
         C = add(X,Y,Z)  ; C = [X,Y] ; C = mult(X,Y) ;
         C = minus(X,Y)),(Var == X ; Var == Y ; Var == Z)) -> true
                        ;
         member_cons(Var,Cs).

%------------------------------%
%  Heuristics Symptom Analyzer  %
%------------------------------%

heu_symp_analyzer(ID,Symptom) :-
        heuristics([Symptom],Fault),
        insert(Fault,ID).

%---------------------------%
%  Process Symptom Analyzer  %
%---------------------------%

pro_symp_analyzer(ID,Symptom) :-
        process(Symptom,individuals(Ilist),initiating_conditions(IClist),
                sustaining_conditions(SClist),_),
        (Symptom =.. [Proc,Arg1,Arg2,Dev];
         Symptom =.. [Proc,Arg1,Dev]),
        (pro_dev(Symptom,Dev) -> pro_symp_analyzer1(Symptom,ID,Dev,SClist)
                              ;
                              pro_symp_analyzer2(Symptom,ID,Ilist,IClist,SClist)).

pro_dev(Symptom,Dev) :-
        device(Dev,_,processes(Plist),_,_),
        member(Symptom,Plist),!.

pro_symp_analyzer1(Symptom,ID,Dev,SClist) :-
        p_create_new_symptoms(SClist,Faults,Dev),
        insert(Faults,ID).

p_create_new_symptoms([],[],_) :- !.
p_create_new_symptoms([SC|SClist],Fault,Dev) :-
        (SC = equal(X,Y) -> Fault1 = [[Dev,X,+],[Dev,X,-],[Dev,Y,+],[Dev,Y,-]] ;
```

```
        SC = greater-than(X,Y) -> Fault1 = [[Dev,X,+],[Dev,Y,-]] ;
        SC = less-than(X,Y) -> Fault1 = [[Dev,X,-],[Dev,Y,+]] ;
        SC = [X,Y] -> Fault1 = [[Dev,X,+]]),
        p_create_new_symptoms(SClist,Fault2,Dev),
        append(Fault1,Fault2,Fault).

pro_symp_analyzer2(Symptom,ID,Ilist,IClist,SClist) :-
        ((Symptom =.. [Proc,Arg1,Arg2,Dev], nonvar(Arg1), nonvar(Arg2)) ;
        (Symptom =.. [Proc,Arg1,Dev], nonvar(Arg1))) -> insert(Ilist,ID),
                                                        insert(IClist,ID),
                                                        insert(SClist,ID)
                        ;
        structural_fls1(Symptom,ID).

pro_symp_analyzer(ID,not_active(Proc)) :-
        process(Proc,individuals(Ilist),initiating_conditions(IClist),
                sustaining_conditions(SClist),_),
        negateindiv(Ilist,ID),
        negateinit(IClist,ID),
        negatesus(SClist,ID).

negateindiv([],_) :- !.
negateindiv([X|Ts],ID):-
        insert(no(X),ID),
        negateindiv(Ts,ID).

negateinit([],_) :- !.
negateinit([X|Ts],ID) :-
        (process(X,_,_,_,_) -> insert(not_active(X),ID)
                        ;
        naive_physics([X],_) -> insert(not_(X),ID)
                        ;
        insert(no(X))),
        negateinit(Ts,ID).

negatesus([],_) :- !.
negatesus([X|Ts],ID) :-
        (process(X,_,_,_,_) -> insert(not_active(X),ID)
                        ;
         naive_physics([X],_) -> insert(not_(X),ID)
                        ;
         X = greater_than(A1,A2) -> insert([[A1,-],[A2,+]],ID)
                        ;
         X = less_than(A1,A2) -> insert([[A1,+],[A2,-]],ID)
                        ;
```

```
        X = equal(A1,A2) -> insert([[A1,-],[A1,+],[A2,+],[A2,-]],ID)
                          ;
        insert(no(X))),
        negatesus(Ts,ID).

%----------------------------------%
%  Naive Physics Symptom Analyzer  %
%----------------------------------%

naive_phy_symp_analyzer(ID,Symptom) :-
        naive_physics([Symptom],Faults),
        insert(Faults,ID).



%  Miscellaneous  %

insert([],_):- !.
insert(Fault,ID):-
        notlist(Fault),
        ifvar(Fault,Fault1),
        (call(f_hypothesis(_,_,Fault1)) -> true
                ;
        getID(FaultID),
        assertz(f_hypothesis(FaultID,ID,Fault)),
        assertz(active(FaultID)),!).
insert([F|Fault],ID):-
        ifvar(F,F1),
        (call(f_hypothesis(_,_,F1)) -> true
                ;
        getID(FaultID),
        assertz(f_hypothesis(FaultID,ID,F)),
        assertz(active(FaultID))),
        insert(Fault,ID).

ifvar(X,Y) :-
        var(X),
        Y = flag,!.
ifvar(X,X).

getID(ID) :-
        retract(id(N)),
        ID is N + 1,
        assert(id(ID)).
```

```
copy(A,B) :-
        assert(cp(A)),
        retract(cp(B)).

member(X,[X|_]).
member(X,[Y|Z]) :- member(X,Z).

append([],X,X).
append([X|Xs],Y,[X|Zs]) :- append(Xs,Y,Zs).

notlist([X|Y]) :- !,false.
notlist(X).
```

Appendix B

Diagnostic Strategies

This appendix gives concise implementations of the applied diagnostic strategies. For simplicity, the qualitative value propagation and direct path of causality are incorporated.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Qualitative Value Propagation  %
%                &                 %
%     Direct Path of Causality     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

qvprop_dirpath([Dev,Var,Val],ID) :-
        device(Dev,_,_,terminal(Input,_),constraints(Cs)),
        Input = [input_1(connection([Comp]),parameters(Plist))],
        Cs = [state_1(S_cond,Constraints)],
        make_par_val_list(Plist,Plistval),
        propagate([Var,Val],Plistval,Constraints),
        device(Comp,_,_,terminal(_,Output),_),
        Output = [output_1(_,parameters(Complist))],
        make_par_val_list(Complist,Complistval),
        set_value(Plistval,Complistval),
        get_instval_only(Complistval,Complistval2),
        set_device(Comp,Complistval2,Newsymptoms),
        insert(Newsymptoms,ID).

make_par_val_list([],[]) :- !.
make_par_val_list([Var|Vars],[[Var,Val]|Varvals]) :-
        make_par_val_list(Vars,Varvals).

propagate([Var,Val],Plistval,Constraints):-
        Varlist = [[Var,Val]|Plistval],
        length(Constraints,LCS),
        loopprop(Varlist,Constraints,LCS).
```

```
loopprop(_,_,0) :- !.
loopprop(S,Constraints,N) :-
        propagate2(S,Constraints),
        N1 is N -1,
        loopprop(S,Constraints,N1).

propagate2(_,[]) :- !.
propagate2(S,[C|Constraints]) :-
        propagate2_1(S,C),
        propagate2(S,Constraints).

propagate2_1(S,m_plus(X,Y)) :-
        member([X,Xval],S),
        member([Y,Yval],S),
        (check_var(Xval,Yval) -> true
                ;
         check_one_var(Xval,Yval) -> m_plus_tab(Xval,Yval)
                ;
         true).
propagate2_1(S,m_minus(X,Y)) :-
        member([X,Xval],S),
        member([Y,Yval],S),
        (check_var(Xval,Yval) -> true
                ;
         check_one_var(Xval,Yval) -> m_minus_tab(Xval,Yval)
                ;
         true).
propagate2_1(_,_) :- !.

check_var(X,Y) :-
        var(X),!,var(Y),!.

check_one_var(X,Y) :-
        (var(X),nonvar(Y)) -> true
                ;
        (var(Y),nonvar(X)).

m_plus_tab(0,0).
m_plus_tab(+,+).
m_plus_tab(-,-).

m_minus_tab(0,0).
m_minus_tab(+,-).
m_minus_tab(-,+).
```

```
set_value([],_) :- !.
set_value([[Var,Val]|Varvals],Complist) :-
        nonvar(Val),
        Var =.. [Functor|Args],
        name(Functor,[105,110|Ls]),
        name(Functor1,[111,117,116|Ls]),
        Var1 =.. [Functor1|Args],
        member([Var1,Val],Complist),
        set_value(Varvals,Complist).
set_value([H|T],Complist) :-
        set_value(T,Complist).

get_instval_only([],[]) :- !.
get_instval_only([[Var,Val]|Cs],[[Var,Val]|Cs2]) :-
        nonvar(Val),
        get_instval_only(Cs,Cs2).
get_instval_only([[Var,Val]|Cs],Cs2) :-
        get_instval_only(Cs,Cs2).

set_device(_,[],[]) :- !.
set_device(Dev,[[Var,Val]|Varvals],[[Dev,Var,Val]|Dvvs]) :-
        set_device(Dev,Varvals,Dvvs).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Structural Fault Localization  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

structural_fls1(Symptom,ID) :-
        Symptom =.. [Proc,Subs,Lsrc,Dst],
        search_trp(Dst,Complist),
        is_there(Subs,Complist,CL),
        not_connected(Dst,CL,Lsrc1),
        (Lsrc1 = [[Lsrc]] ->
                process(Symptom,individuals(Ilist),initiating_conditions(IClist),
                        sustaining_conditions(SClist),_),
                insert(Ilist,ID),
                insert(IClist,ID),
                insert(SClist,ID)
                        ;
         Lsrc1 = [Cp|Cps] ->
                c_symptoms_for_each_compo(Lsrc1,Symptom,ID)
                        ;
         true).
```

```
c_symptoms_for_each_compo([],_,_) :- !.
c_symptoms_for_each_compo([Cp|CPs],Symptom,ID) :-
        copy(Symptom,Symptom1),
        Symptom =.. [Proc,Subs,Cp,Dst],
        process(Symptom,individuals(Ilist),initiating_conditions(IClist),
                sustaining_conditions(SClist),_),
        insert(Ilist,ID),
        insert(IClist,ID),
        insert(SClist,ID),
        c_symptoms_for_each_compo(CPs,Symptom1,ID).

search_trp(Dst,Complist) :-
        topological_relative_position(TrpName,inside(Complist)),
        member(Dst,Complist).

is_there(Subs,[],[]):- !.
is_there(Subs,[C|Complist],[C|Cs]):-
        device(C,_,_,terminal([input_1(_,parameters(Parlist))],_),_),
        is_subs_in_parlist(Subs,Parlist),
        is_there(Subs,Complist,Cs).
is_there(Subs,[C|Complist],Cs) :-
        is_there(Subs,Complist,Cs).

is_subs_in_parlist(Subs,[]) :- !,fail.
is_subs_in_parlist(Subs,[P|Ps]) :-
        P =.. [Func,Arg],
        Subs == Arg -> true
                ;
        is_subs_in_parlist(Subs,Ps).

not_connected(Dst,[],[]):-!.
not_connected(Dst,[C|Cls],[C|Lsrc]) :-
        device(C,_,_,terminal([input_1(connection(ICL),_)],
                              [output_1(connection(OCL),_)]),_),
        not(member(Dst,ICL)),
        not(memeber(Dst,OCL)),
        not_connected(Dst,Cls,Lsrc).
not_connected(Dst,[C|Cls],Lsrc) :-
        not_connected(Dst,Cls,LSrc).
```

Appendix C

Sorts of Knowledge

This appendix contains the domain model and additional domain knowledge on the domain of elementary refrigeration plants. All of them are written in terms of *facts* statements in Prolog.

```
%%%%%%%%%%%%%%%%%%%%%%%%
%%                      %%
%%    DOMAIN MODEL      %%
%%                      %%
%%%%%%%%%%%%%%%%%%%%%%%%

%--------------------%
%    Device Model    %
%--------------------%

device(compressor,
  purpose([state_1(S_Cond,[outpress(gas),+])]),
  processes([compression(gas,compressor)]),
  terminal([input_1(connection([conduit_4]),
             parameters([inflow(gas),inpress(gas),intemp(gas)]))],
           [output_1(connection([conduit_1]),
             parameters([outflow(gas),outpress(gas),outtemp(gas)]))]),
 constraints([state_1(S_Cond,[add(inpress(gas),addpress,outpress(gas)),
                      [addpress,+],m_plus(outpress(gas),inpress(gas)),
                      m_plus(outpress(gas),outtemp(gas)),
                      m_plus(outtemp(gas),intemp(gas)),
                      m_plus(inpress(gas),intemp(gas)),
                      m_plus(inflow(gas),outflow(gas))])])).

device(evaporator,
  purpose([state_1(S_Cond,[outflow(gas),+])]),
  processes([flowing(heat,vaporizing_water_system,evaporator)]),
```

```
  terminal([input_1(connection([conduit_3]),
            parameters([inflow(liquid),inpress(liquid),intemp(liquid)]))],
           [output_1(connection([conduit_4]),
            parameters([outflow(gas),outpress(gas),outtemp(gas)]))]),
  constraints([state_1(S_Cond,[m_plus(outflow(gas),inflow(liquid)),
                       m_plus(inflow(liquid),intemp(liquid)),
                       m_minus(outtemp(gas),intemp(liquid)),
                       m_minus(outpress(gas),inpress(liquid)),
                       m_plus(outflow(gas),outtemp(gas))])])).

device(condenser,
  purpose([state_1(S_Cond,[outflow(liquid),+])]),
  processes([flowing(heat,condenser,cooling_water_system)]),
  terminal([input_1(connection([conduit_1]),
            parameters([inflow(gas),inpress(gas),intemp(gas)]))],
           [output_1(connection([conduit_2]),
            parameters([outflow(liquid),outpress(liquid),outtemp(liquid)]))]),
  constraints([state_1(S_Cond,[m_plus(outflow(liquid),inflow(gas)),
                       m_plus(inflow(gas),intemp(gas)),
                       m_plus(outtemp(liquid),intemp(gas)),
                       m_plus(outpress(liquid),inpress(gas)),
                       m_plus(outflow(gas),outtemp(gas))])])).

device(expansion_valve,
  purpose([state_1(S_Cond,[outpress(liquid),-])]),
  processes([expansion(liquid,expansion_valve)]),
  terminal([input_1(connection([conduit_2]),
            parameters([inflow(liquid),inpress(liquid),intemp(liquid)]))],
  [output_1(connection([conduit_3]),
            parameters([outflow(liquid),outpress(liquid),outtemp(liquid)]))]),
  constraints([state_1(S_Cond,[m_plus(outflow(liquid),inflow(liquid)),
                       m_plus(inflow(liquid),inpress(liquid)),
                       m_plus(outpress(liquid),inpress(liquid)),
                       m_plus(outtemp(liquid),intemp(liquid)),
                       m_minus(outpress(liquid),outflow(liquid))])])).

device(conduit_2,
  purpose([state_1(S_Cond,[outflow(liquid),+])]),
  processes([flowing(liquid,condenser,expansion_valve)]),
  terminal([input_1(connection([condenser]),
            parameters([inflow(liquid),inpress(liquid),intemp(liquid)]))],
           [output_1(connection([expansion_valve]),
            parameters([outflow(liquid),outpress(liquid),outtemp(liquid)]))]),
  constraints([state_1(S_Cond,[m_plus(inpress(liquid),outpress(liquid)),
                       m_plus(inpress(liquid),intemp(liquid)),
```

```
                        m_plus(outpress(liquid),outtemp(liquid)),
                        m_plus(inflow(liquid),outflow(liquid)),
                        m_plus(intemp(liquid),outtemp(liquid))])])).

device(conduit_1,
  purpose([state_1(S_Cond,[outflow(gas),+])]),
  processes([flowing(gas,compressor,condenser)]),
  terminal([input_1(connection([compressor]),
            parameters([inflow(gas),inpress(gas),intemp(gas)]))],
           [output_1(connection([condenser]),
            parameters([outflow(gas),outpress(gas),outtemp(gas)]))]),
  constraints([state_1(S_Cond,[m_plus(inpress(gas),outpress(gas)),
                        m_plus(inpress(gas),intemp(gas)),
                        m_plus(outpress(gas),outtemp(gas)),
                        m_plus(inflow(gas),outflow(gas)),
                        m_plus(intemp(gas),outtemp(gas))])])).

device(conduit_3,
  purpose([state_1(S_Cond,[outflow(liquid),+])]),
  processes([flowing(liquid,expansion_valve,evaporator)]),
  terminal([input_1(connection([expansion_valve]),
             parameters([inflow(liquid),inpress(liquid),intemp(liquid)]))],
  [output_1(connection([evaporator]),
            parameters([outflow(liquid),outpress(liquid),outtemp(liquid)]))]),
  constraints([state_1(S_Cond,[m_plus(inpress(liquid),outpress(liquid)),
                        m_plus(inpress(liquid),intemp(liquid)),
                        m_plus(outpress(liquid),outtemp(liquid)),
                        m_plus(inflow(liquid),outflow(liquid)),
                        m_plus(intemp(liquid),outtemp(liquid))])])).

device(conduit_4,
  purpose([state_1(S_Cond,[outflow(gas),+])]),
  processes([flowing(gas,evaporator,compressor)]),
  terminal([input_1(connection([evaporator]),
            parameters([inflow(gas),inpress(gas),intemp(gas)]))],
  [output_1(connection([compressor]),
            parameters([outflow(gas),outpress(gas),outtemp(gas)]))]),
  constraints([state_1(S_Cond,[m_plus(inpress(gas),outpress(gas)),
                        m_plus(inpress(gas),intemp(gas)),
                        m_plus(outpress(gas),outtemp(gas)),
                        m_plus(inflow(gas),outflow(gas)),
                        m_plus(intemp(gas),outtemp(gas))])])).

%--------------------%
%    Process Model   %
```

```
%--------------------%

process(compression(Subs,X),
  individuals([Subs,X]),
  initiating_conditions([exist(Subs,X)]),
  sustaining_conditions([[addpress(X),+]]),
  effects([[press(Subs),+]])).

process(flowing(liquid,Src,Dst),
  individuals([contained_liquid(Src),Dst]),
  initiating_conditions([aligned(liquid_flow_path,Src,Dst)]),
  sustaining_conditions([greater_than(press(Src),press(Dst))]),
  effects([inc(amount_of(liquid,Dst)),dec(amount_of(liquid,Src)),
           equal(press(Src),press(Dst))])).

process(flowing(heat,Src,Dst),
  individuals([contained_heat(Src),Dst]),
  initiating_conditions([aligned(heat_flow_path,Src,Dst)]),
  sustaining_conditions([greater_than(temp(Src),temp(Dst))]),
  effects([inc(amount_of(heat,Dst)),dec(amount_of(heat,Src)),
           equal(temp(Src),temp(Dst))])).

process(expansion(Subs,X),
  individuals([Subs,X]),
  initiating_conditions([exist(Subs,X)]),
  sustaining_conditions([[addpress(X),-]]),
  effects([[press(Subs),-]])).

process(melting(solid,X),
  individuals([solid,X]),
  initiating_conditions([flowing(heat,Src,X)]),
  sustaining_conditions([greater_than(temp(solid),t-melt(solid))]),
  effects([inc(amount_of(liquid,X)),dec(amount_of(solid,X))])).

process(condensation(gas,X),
  individuals([gas,X]),
  initiating_conditions([exist(gas,X)]),
  sustaining_conditions([flowing(heat,X,Dst)]),
  effects([inc(amount_of(liquid,X)),dec(amount_of(gas,X)),
           dec(amount-of(heat,X))])).

process(flowing(gas,Src,Dst),
  individuals([contained_gas(Src),Dst]),
  initiating_conditions([aligned(gas_flow_path,Src,Dst)]),
  sustaining_conditions([greater_than(press(Src),press(Dst))]),
```

```
  effects([inc(amount_of(gas,Dst)),dec(amount_of(gas,Src)),
                equal(press(Src),press(Dst))])).

process(sublimation(solid,X),
  individuals([solid,X]),
  initiating_conditions([exist(gas,X)]),
  sustaining_conditions([flowing(heat,Src,X)]),
  effects([inc(amount_of(gas,X)),dec(amount_of(solid,X)),
           inc(amount-of(heat,X))])).

process(evaporation(liquid,X),
  individuals([liquid,X]),
  initiating_conditions([exist(liquid,X)]),
  sustaining_conditions([boiling(liquid,X)]),
  effects([inc(amount_of(gas,X)),dec(amount_of(liquid,X)),
           dec(amount-of(heat,X))])).

process(boling(liquid,X),
   individuals([liquid,X]),
   initiating_conditions([flowing(heat,Src,X)]),
   sustaining_conditions([greater-than(temp(liquid),t-boil(liquid))]),
   effects([inc(amount_of(gas,X)),dec(amount_of(liquid,X))])).

process(moving(solid,Src,Dst),
   individuals([contained_solid(Src),Dst]),
   initiating_conditions([aligned(solid_flow_path,Src,Dst)]),
   sustaining_conditions([greater_than(net_force(solid),ZERO)]),
   effects([inc(amount_of(solid,Dst)),dec(amount_of(solid,Src)),
            equal(net_force(solid),ZERO)])).

process(freezing(liquid,X),
   individuals([liquid,X]),
   initiating_conditions([flowing(heat,X,Src)]),
   sustaining_conditions([lower-than(temp(liquid),t-freezing(liquid))]),
   effects([inc(amount_of(solid,X)),dec(amount_of(liquid,X))])).

process(condensation(gas,X),
   individuals([gas,X]),
   initiating_conditions([exist(gas,X)]),
   sustaining_conditions([flowing(heat,X,Dst)]),
   effects([inc(amount_of(solid,X)),dec(amount_of(gas,X)),
            dec(amount-of(heat,X))])).

%--------------------------------%
%  Topological Relative Position  %
```

```
%---------------------------------%

topological_relative_position(outdoor_unit,
   inside([compressor,propeller_fan,conduit_1,conduit_2,conduit_4])).

topological_relative_position(indoor_unit,
   inside([condenser,expansion_valve,conduit_3,evaporator])).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                                        %%
%%    ADDITIONAL DOMAIN KNOWLEDGE         %%
%%                                        %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%--------------%
% Heuristics  %
%--------------%

heuristics([knocking(compressor)],
           [compression(liquid,compressor)]).

%-----------------%
% Naive Physics  %
%-----------------%

naive_physics([exist(liquid,X)],
              [flowing(liquid,Src,X),melting(solid,X),
               condensation(gas,X)]).

naive_physics([exist(gas,X)],
              [flowing(gas,Src,X),sublimation(solid,X),
               evaporation(liquid,X)]).

naive_physics([exist(solid,X)],
              [moving(solid,Src,X),freezing(liquid,X),
               condensation(gas,X)]).


naive_physics([not_(exist(liquid,X))],
              [flowing(liquid,X,Dst),freezing(liquid,X),
               evaporation(liquid,X)]).

naive_physics([not_(exist(gas,X))],
              [flowing(gas,X,Dst),condensation(gas,X)]).
```

```
naive_physics([not_(exist(solid,X))],
              [moving(solid,X,Dst),sublimation(solid,X),
               melting(solid,X)]).
```

Bibliography

[1] Abbott, K. H. (1988). Robust Operative Diagnosis as Problem Solving in a Hypothesis Space. *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Saint Paul, Minnesota, 369–374.

[2] Abu-Hanna, A., & Benjamins, V.R. (1990). Knowledge Classification of Models in Model-Based Diagnosis, *The Tenth International Workshop on Expert Systems and Their Applications, General Conference on Second Generation Expert Systems*, Avignon, France, 97–110.

[3] Ajjanagadde, V. (1993). Icorporating Background Knowledge and Structured Explananda in Abductive Reasoning: A Framework. *IEEE Transactions on Systems, Man, and Cybernetics*, 23, 3, 650–664.

[4] Ayeb, B. E. (1994). On the Applicability of Model-Based Diagnosis: Examining the Case of Multiple Observations. *Proceedings of the Seventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-94)*, Austin, Texas, 435–446.

[5] Boden, M. A. (1990). *The Philosophy of Artificial Intelligence.* New York: Oxford University Press.

[6] Bonarini, P., & Sassaroli, P. (1991). *Oppurtunistic Multimodel-Based Diagnosis: using all the knowledge we have* (Report No. 91-036). Dipartimento di Elettronica, Politecnico di Milano, Milano, Italy.

[7] Bonarini, P., & Sassaroli, P. (1995). Uncertainty and Approximation in Multimodel-based Industrial Diagnosis. *Working Papers of the Sixth International Workshop on Principles of Diagnosis (DX-95)*, Göslar, Germany, 8–13.

[8] Bond, G.W. (1994). *Logic Programs for Consistency-Based Diagnosis.* PhD thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada.

[9] Bötcher, C. (1995) No Faults in Structure? - How to Diagnose Hidden Interactions. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montréal, Canada, 2, 1728–1734.

[10] Bratko, I., Mozetič, I., & Lavrač, N. (1989). *KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems.* Massachusetts: The MIT Press.

[11] Brown, J.S., Burton, R.R., & de Kleer, J. (1982). Pedagogical, Natural Language, and Knowledge Engineering Techniques in SOPHIE I, II, III. In D.Sleeman & J.S.Brown (Eds.), *Intelligent Tutoring Systems.* New York: Academic Press.

[12] Breuer, M.A., & Friedman, A. (1976). *A Diagnosis and Reliable Design of Digital Systems.* Rocville, MD: Computer Science Press.

[13] Cerepnalkovski, I. (1991). *Modern Refrigerating Machines.* Netherlands: Elsevier Science Publishers B.V.

[14] Chandrasekaran, B. (1987). Data Validation During Diagnosis, A Step Beyond Traditional Sensor Validation. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle, Washington, 778–782.

[15] Chien, S., Doyle, R., & Rouquette, N. (1991). Sensor Placement for Diagnosability in Space-borne Systems: A Model-based Reasoning Approach. *Working Papers of the 2nd International Workshop on the Principles of Diagnosis*, Milan, Italy.

[16] Chin, H., & Danai, K. (1993). Improved Flagging for Pattern Classiying Diagnostic Systems. *IEEE Transaction on Systems, Man, and Cybernetics*, 23, 4, 1101–1107.

[17] Chittaro, L., Guida, G., Tasso, C., & Toppano, E. (1993). Functional and Teleological Knowledge in the Multimodelling Approach for Reasoning About Physical Systems: A Case Study in Diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics*, 23, 6, 1718–1751.

[18] Chu, B. B. (1993). Fault Diagnosis with Continuous System Models. *IEEE Transactions on Systems, Man, and Cybernetics*, 23, 1, 55–64.

[19] Cunningham, P., & Brady, M. (1987). Qualitative Reasoning in Electronic Fault Diagnosis. *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milan, Italy, 1, 443–445.

[20] Dague, P., Raiman, O., & Devès, P. (1987). Troubleshooting: when modeling is the trouble. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle, Washington, 600–605.

[21] Davis, E. (1990). *Representations of Commonsense Knowledge.* San Mateo, CA: Morgan Kaufmann Publishers, Inc.

[22] Davis, R. (1982, Spring). Where Are We ? And Where Do We Go From Here ? *AI Magazine, 3, 1,* 23–24.

[23] Davis, R. (1984). Diagnostic Reasoning Based on Structure and Bahavior. *Artificial Intelligence, 24,* 347–410.

[24] Davis, R., & Hamscher, H. (1988). Model-Based Reasoning: Troubleshooting. In H.E. Shrobe (Ed.), *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence.* San Mateo, California: Morgan Kaufmann Publishers, Inc.

[25] de Kleer, J., & Brown, J.S. (1984). A Qualitative Physics Based on Confluences. *Artificial Intelligence, 24,* 7–83.

[26] de Kleer, J. (1986). An Assumption-Based TMS. *Artificial Intelligence, 28,* 127–162.

[27] de Kleer, J., & Williams, B.C. (1987). Diagnosing Multiple Faults. *Artificial Intelligence, 32,* 97–130.

[28] de Kleer, J., & Williams, B. C. (1991). Special Volume Qualitative Reasoning About Physical Systems II. *Artificial Intelligence,* 51.

[29] de Kleer, J., Mackworth, A.K., & Reiter, R. (1992). Charaterizing Diagnoses and Systems. *Artificial Intelligence, 56,* 197–222.

[30] Downing, K. L. (1987). Diagnostic Improvement Through Qualitative Sensitivity Analysis and Aggregation. *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87),* Seattle, Washington, 789–793.

[31] Dvorak, D.L. (1992). *Monitoring and Diagnosis of Continuous Dynamic Systems Using Semiquantitative Simulation.* PhD thesis, Dept. of Computer Sciences, The University of Texas at Austin, TX.

[32] Ernst, G., & Newell, A. (1969). *GPS: A case study in generality and problem solving.* New York: Academic Press.

[33] Far, B. H., Nakamichi, M. (1993). Qualitative Fault Diagnosis in Systems with Nonintermitten Concurrent Faults: A Subjective Approach. *IEEE Transactions on Systems, Man, and Cybernetics*, 23, 1, 14–30.

[34] Fesq, L. M., & McNamee L. P. (1994). Does Observability Imply Diagnosability? *Proceedings of the Seventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-94)*, Austin, Texas, 649–656.

[35] Fischler, M. A., & Firschein, O. (1987). *Intelligence: The Eye the Brain, and the Computer*, Massachusetts: Addison-Wesley Publishing Company.

[36] Fishwick, P. A., Narayanan, H. N., Sticklen, J., & Bonarini, A. (1994). A Multimodel Approach to Reasoning and Simulation. *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 10, 1433–1449.

[37] Forbus, K.D. (1984). Qualitative Process Theory. *Artificial Intelligence, 24*, 85–168.

[38] Freitag, H. (1990). A Generic Measurements Proposer. *Proceedings of International Workshop on Expert Systems in Engineering*, Vienna, Austria.

[39] Freitag, H. (1991). Goal-Driven Structural Focusing in Model-Based Diagnosis. *Working Papers of the 2nd International Workshop on Principle of Diagnosis (DX-91)*, Milan, Italy.

[40] Friedrich, G., Gottlob, G., & Nejdl, W. (1990). Physical Impossibility Instead of Fault Models. *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, Boston, 331–336.

[41] Friedrich, G., & Lackinger, F. (1991). Diagnosing Temporal Misbehavior. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, 1116–1122.

[42] Genesereth, M.R. (1984). The Use of Design Descriptions in Automated Diagnosis. *Artificial Intelligence, 24*, 411–436.

[43] Guan, J., & Graham, J. H. (1994). Diagnostic Reasoning With Fault Propagation Digraph and Sequential Testing. *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 10, 1552–1558.

[44] Hamscher, W., Console, L., & de Kleer, J. (1992). *Readings in Model-Based Diagnosis.* San Mateo, CA: Morgan Kaufmann Publishers, Inc.

[45] Hart, P. (1982). Directions for AI in the eighties. *SIGART Newsletter, 79*, 11–16.

[46] Hayes, P.J. (1979). The Naive Physics Manivesto. In D. Michie (Ed.), *Expert Systems in the Micro-Electronic Age.* Edinburgh: Edinburgh University Press.

[47] Hayes-Roth, F., Waterman, D. A., & Lenat, D. B. (1983). *Building Expert Sytems.* Reading, MA: Addison-Wesley Publishing Company.

[48] Hunt, J. (1991). *A Task Specific Integration Architecture for Multiple Problem Solver, Model-Based, Diagnostic Expert Systems.* PhD thesis, Dept. of Computer Science, University of Wales Aberystwyth, UK.

[49] Ishida, Y., Adachi, N., & Tokumaru, H. (1985). A Topological Approach to Failure Diagnosis of Large-Scale Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 15, 3, 327–333.

[50] Kuipers, B. (1984). Commonsense reasoning about causality: Deriving behavior from Structure. *Artificial Intelligence, 24*, 169–204.

[51] Kuipers, B. (1986). Qualitative Simulation. *Artificial Intelligence, 29*, 289–338.

[52] Lackinger, F. (1991). *Model-Based Troubleshooting: Qualitative Reasoning and the Impacts of Time.* PhD thesis, Dept. of Computer Science, the Vienna University of Technology, Austria.

[53] Lee, S. C. (1994). Sensor Value Validation Based on Systematic Exploration of the Sensor Redundancy for Fault Diagnosis KBS. *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 4, 594–605.

[54] Lindsay, R.K., Buchanan, B.G., Feigenbaum, E.A., & Lederberg, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project.* New York: McGraw-Hill.

[55] Merritt, D. (1989). *Building Expert Systems in Prolog.* New Yor: Springer-Verlag.

[56] Michie, D. (1982, Spring). High-Road and Low-Road Programs. *AI Magazine, 3, 1*, 21–22.

[57] Nihonreitoukyoukai, *A Technical Guide to Air Conditioning and Refrigerating Systems: A standard texbook for beginners.* (Reitoukuuchougijutsu - Shoukyuu hyoujun tekisuto). Tokyo: Nihonreitoukyoukai. *(In Japanese).*

[58] Ousumi, K. (1977). *A Practical Guide for diagnosing faults in refrigeration plants.* (Reitouki koshou shuuri no jissai). Tokyo: Ohm Publishers, Inc. *(In Japanese).*

[59] Pan, Y. C. (1984a). *Qualitative Reasonings with Deep-Level Mechanism Models for Diagnoses of Dependent Failures.* PhD Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Illinois.

[60] Pan, Y. C. (1984b). Qualitative Reasonings with Deep-Level Mechanism Models for Diagnoses of Mechanism Failures. *Proceedings of the First IEEE Conference on Artificial Intelligence (CAAI-84)*, Denver, CO, 295–301.

[61] Pearce, D. A. (1988). The Induction of Fault Diagnosis Systems from Qualitative Models. *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Saint Paul, Minnesota, 353–357.

[62] Poole, D. (1988). Representing Knowledge for Logic-based Diagnosis. *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, Tokyo, Japan, 1282–1290.

[63] Preist, C., & Welham, B. (1990). Modelling Bridge Faults for Diagnosis in Electronic Circuits. *Working Papers of The First International Workshop on Principles of Diagnosis (DX-90)*, Stanford, 69–73.

[64] Purna, Y.W., & Yamaguchi, T. (1993a). A Framework for Hypothesis Generation in Model-Based Diagnosis. *Proceedings of '93 Korea/Japan Joint Conference on Expert Systems*, Seoul, Korea, 144–159.

[65] Purna, Y.W., & Yamaguchi, T. (1993b). Diagnosing Faults Using Useful Sorts of Knowledge and Strategies. *Proceedings of The Sixth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-93)*, Edinburgh, Scotland, 299–308.

[66] Purna, Y.W., & Yamaguchi, T. (1993c). Exploiting Multifarious Knowledge and Strategies for Diagnosing Faults: An Alternative to Model-Based Diagnosis. *Proceedings of The Third International Conference for Young Computer Scientists (ICYCS'93)*, Beijing, China, 2.37–2.40.

[67] Purna, Y.W. (1994a). A Framework for Hypothesis Generation in Model-Based Diagnosis. *Reports of The Graduate School of Electronic Science and Technology Shizuoka University*, 15, 133–146.

[68] Purna, Y.W., & Yamaguchi, T. (1994b). Hypothesis Generation and Testing in MODEST. *Proceedings of the 3rd Pacific Rim International Conference on Artificial Intelligence (PRICAI-94)*, Beijing, China, 1, 106–112.

[69] Purna, Y.W., & Yamaguchi, T. (1995a). A Framework for Hypothesis Generation in Model-Based Diagnosis. *Expert Systems with Applications: An International Journal*, 9, 1, 41–54.

[70] Purna, Y.W., & Yamaguchi, T. (1995b). MODEST: A Framework for Diagnostic Expert Systems (Preliminary Result). *Working Papers of The 6th International Workshop on Principles of Diagnosis (DX-95)*, Göslar, Germany, October, 86–90.

[71] Purna, Y.W., & Yamaguchi, T. (1996). Generating and Testing Fault Hypotheses with MODEST. to appear in *Proceedings of the 3rd World Congress on Expert Systems (WCES-3)*, Seoul, Korea, February, 1996.

[72] Ramamurthi, K., & Hough, Jr., C. L. (1994). A Hybrid Approach for Robust Diagnostic of Cutting Tools. *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 3, 482–492.

[73] Rameshi, T. S., Shum, S. K., & Davis, J. F. (1988). A Structured Framework for Efficient Problem Solving in Diagnostic Expert Systems. *Computers & Chemical Engineering*, 12, 9/10, 891–902.

[74] *Random House Webster's College Dictionary.* (1990). New York: Random House, Inc.

[75] Rasmussen, J. (1993). Diagnostic Reasoning in Action. *IEEE Transactions on Systems, Man, and Cybernetics*, 23, 4, 981–991.

[76] Reiter, R. (1987). A Theory of Diagnosis from First Principles. *Artificial Intelligence, 32*, 57–96.

[77] Reed, N. E., Stuck, E. R., & Moen, J. B. (1988). Specialized Strategies: An Alternative to First Priciples in Diagnostic Problem Solving. *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, Saint Paul, Minnesota, 364–368.

[78] Rich, E. (1983). *Artificial Intelligence.* New York: McGraw-Hill.

[79] Rychener, M. D. (1988). *Expert Systems for Engineering Design.* San Diego, CA: Academic Press, Inc.

[80] Sembugamoorthy, V., & Chandrasekaran B. (1986). Functional Representation of Devices and Compilation of Diagnostic Problem-Solving. In J. Kolodner & C. Riesbeck (Eds.), *Experience, Memory and Learning.* London: Lawrence Erlbaum & Associates.

[81] Shearer, J. L., Murphy, A. T., & Richardson, H. H. (1967). *Introduction to System Dynamics*, Massachusetts: Addison-Wesley Publishing Company.

[82] Shiozaki, J., Matsuyama, H., O'Shima, E., & Iri, M. (1985). An Improved Algorithm for Diagnosis of System Failures in the Chemical Process. *Computers & Chemical Engineering*, 9, 3, 285–293.

[83] Shirley, M.H., & Davis, R. (1983). Generating Distinguishing Tests based on Hierachical Models and Symptom Information. *IEEE International Conference on Computer Design.*

[84] Shortliffe, E.H. (1976). *Computer-Based Medical Consultations: MYCIN* New York: American Elsevier.

[85] Shum, S.K., Davis, J.F., Punch III, W.F., & Chandrasekaran, B. (1988). An Expert System Approach to Malfunction Diagnosis in Chemical Plants. *Computers & Chemical Engineering*, 12, 1, 27–36.

[86] Simmons, R., & Davis, R. (1987). Generate, Test and Debug: Combining Associational Rules and Causal Models. *Proceedings of Tenth International Joint Conference on Artificial Intelligence, 2*, 1071–1078. Milano, Italy.

[87] Sonntag, R.E., & Van Wylen G.J. (1971). *Introduction to Thermodynamics: Classical and Statistical.* New York: John Wiley & Sons, Inc.

[88] Sterling, L, & Shapiro, E. (1986). *The Art of Prolog: Advance Programming Techniques.* Cambridge: The MIT Press.

[89] Struss, P. (1987). Multiple Representation of Structure and Function. In J. Gero (Ed.), *Expert Systems in Computer-Aided Design.* Amsterdam: Elsevier Science Publishers.

[90] Struss, P., & Dressler, O. (1988) "Physical Negation" - Integrating Fault Models into the General Diagnostic Engine. *Proceedings of the 11th International Conference on Artificial Intelligence (IJCAI-89)*, Detroit, MI, 1318–1323.

[91] Struss, P. (1990). *Model-Based Diagnosis - Introduction and Survey of Recent Work by ARM.* Siemens Report INF 2 ARM-16-90.

[92] Struss, P. (1995). Qualitative Modeling is the Key: A Successful Feasibility Study in Automated Generation of Diagnosis Guidelines and Failure Mode and Effects Analysis for Mechatronic Car Subsystems. *Working Papers of the Sixth International Workshop on Principles of Diagnosis*, Göslar, Germany, 99–106.

[93] Sueda, N. & Iwamasa, M. (1995). A Pilot System for Plant Control Using Model-Based Reasoning. *IEEE Expert: Intelligent Systems & Their Applications*, 10, 4, 24–31.

[94] Torikoshi, A. (1989). *A Framework for Knowledge Generation Systems Based on Deep Knowledge.* Bachelor's thesis. Shizuoka University, Japan. (*In Japanese*).

[95] Throop, D.R. (1991). *Model-Based Diagnosis of Complex, Continuous Mechanisms.* PhD thesis, Dept. of Computer Sciences, The University of Texas at Austin, TX.

[96] Tsybenko, Y. V. (1995). Model-Based System Decomposition. *Working Papers of the Sixth International Workshop on Principles of Diagnosis*, Göslar, Germany, 115–122.

[97] Van de Velde, W. (1986). Explainable Knowledge Production. *Proceedings of the Seventh European Conference on Artificial Intelligence, 1*, 8–22.

[98] Venkatasubramanian, V., & Rich, S. H. (1988). An Objected-Oriented Two-Tier Architecture for Integrating Compiled and Deep-Level Knowledge for Process Diagnosis. *Computers & Chemical Engineering*, 12, 9/10, 903–921.

[99] 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, 181–189.

[100] Waterman, D. A. (1986). *A Guide to Expert Systems.* Reading, MA: Addison-Wesley Publishing Company.

[101] Weiss, S. M., & Kulikowski C. A. (1984). *Practical Guide to Designing Expert Systems.* Totowa, NJ: Rowman and Allanheld.

[102] Weiss, S. M., & Kulikowski C. A. (1990). *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, San Mateo, CA: Morgan Kaufmann Pub., Inc.

[103] Weld, D. S. & de Kleer, J. (1990). *Readings in Qualitative Reasoning About Physical Systems.* San Mateo, CA: Morgan Kaufmann Publishers, Inc.

[104] Winston, P.H. (1992) *Artificial Intelligence—3rd edition.* New York: Addison-Wesley Publishing Company.

[105] Yamaguchi, T., Mizoguchi, R., Nakamura, H., Ozawa, T., Torikoshi, A., Nomura, Y., & Kakusho, O. (1992). Knowledge Compiler II Based on Domain Model and Failure Model. *Journal of Japanese Society for Artificial Intelligence, 7, 4,* 663–674. (*In Japanese*).

[106] Yeung, R. W. (1994). On Noiseless Diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 7, 1074–1082.

[107] Zadrozny, W. (1990, July). *The Logic of Abduction.* Paper presented at the First International Workshop on Principles of Diagnosis, Standford University.

List of Publications

Journals & International Conferences:

1. Purna, Y. W., & Yamaguchi, T. (1993). A Framework for Hypothesis Generation in Model-Based Diagnosis, *(Short Version). Proceedings of '93 Korea/Japan Joint Conference on Expert Systems (KJJCES-93)*, pp. 144–159.

2. Purna, Y. W., & Yamaguchi, T. (1993). Diagnosing Faults Using Useful Sorts of Knowledge and Strategies. *Proceedings of the 6th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-93)*, pp. 299–308.

3. Purna, Y. W., & Yamaguchi, T. (1993). Exploiting Multifarious Knowledge and Strategies for Diagnosing Faults. *Proceedings of the 3rd International Conference for Young Computer Scientists (ICYCS-93)*, pp. 2.37–2.40.

4. Purna, Y. W. (1994). A Framework for Hypothesis Generation in Model-Based Diagnosis. *Reports of The Graduate School of Electronic Science and Technology Shizuoka University*, No. 15, pp. 133–146.

5. Purna, Y. W., & Yamaguchi, T. (1994). Some Notions on Testing Generated Fault Hypotheses. *Proceedings of the 7th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE-94)*, pp. 447–455.

6. Purna, Y. W., & Yamaguchi, T. (1994). Hypothesis Generation and Testing in MODEST. *Proceedings of the 3rd Pacific Rim International Conference on Artificial Intelligence (PRICAI-94)*, Vol. I, pp. 106–112, 1994.

7. Purna, Y. W., & Yamaguchi, T. (1995). A Framework for Hypothesis Generation in Model-Based Diagnosis *(Long Version)*, *Expert Systems with Applications: An International Journal*, Vol. 9, No. 1, pp. 41–54.

8. Purna, Y. W., & Yamaguchi, T. (1995). MODEST: A Framework for Diagnostic Expert Systems, *Working Papers of the 6th International Workshop on Principles of Diagnosis (DX-95)*, pp. 86-90.

9. Purna, Y. W,, & Yamaguchi, T. (1996). Generating and Testing Fault Hypotheses with MODEST, to appear in *Proceedings of the 3rd World Congress on Expert Systems (WCES-3)*, Seoul, Korea, February.

National Conferences:

1. Purna, Y. W., & Yamaguchi, T. (1993). Towards Identifying Useful Sorts of Knowledge and Strategies for Diagnosing Faults: A Framework for Hypothesis Generation, *Proceedings of the 46th National Conference of Information Processing Society of Japan (IPSJ-46)*, Vol.2, pp. 2-37 – 2-38.

2. Purna, Y. W., & Yamaguchi, T. (1993). A Framework for Hypothesis Generation in Model-Based Diagnosis, *Working Notes of the 25th Workshop of JSAI Special Interest Group on Knowledge Based Systems*, SIG-KBS-9302, pp. 17–24.

3. Purna, Y. W., & Yamaguchi, T. (1993). Exploiting Useful Sorts of Knowledge and Strategies for Fault Diagnosis: A Framework for Hypothesis Generation in Model-Based Diagnosis, *Proceedings of the 7th Annual Conference of Japanese Society for Artificial Intelligence (JSAI-93)*, pp. 619–622.

4. Purna, Y. W., & Yamaguchi, T. (1994). Issues in Testing Fault Hypotheses, Yusuf Wilajati Purna, Takahira Yamaguchi, *Proceedings of the 8th Annual Conference of Japanese Society for Artificial Intelligence (JSAI-94)*, pp. 473–476.

5. Purna, Y. W., & Yamaguchi, T. (1994). On Testing Fault Hypotheses, Yusuf Wilajati Purna, Takahira Yamaguchi, *Technical Report of the Institute of Electronics, Information and Communication Engineers (IEICE)*, Vol. 94, No. 133, pp. 65–72.

6. Purna, Y. W., & Yamaguchi, T. (1995). Diagnosing Faults with MODEST, *Proceedings of the 9th Annual Conference of Japanese Society for Artificial Intelligence (JSAI-95)*, pp. 463–466.

VITA

Yusuf Wilajati Purna was born in Yogyakarta, Indonesia on August 29, 1966, the son of Leonardus and Roberta Subiyat Budiharga. After graduating from De Britto's College (High School) in 1985, he entered Bandung Institute of Technology (ITB) to study Computer Science. However, he stayed only three months in the institute since he joined the Agency of the Assessment and Application of Technology (BPP Teknologi) and received the scholarship from the agency to study abroad. In 1987, he entered Shizuoka University and completed a Bachelor of Engineering degree in Computer Science in 1991. He continued his study to the master course in the same university and graduated in 1993 with a Master of Engineering degree in Computer Science. He began doctoral studies in April 1993 at Graduate School of Electronic Science & Technology, Shizuoka University.

He is a member of American Association for Artificial Intelligence (AAAI), The Institute of Electrical and Electronic Engineers (IEEE), IEEE Computer Society, USENIX, Japanese Society of Artificial Intelligence, (JSAI), and Information Processing Society of Japan (IPSJ).