An integration of case-based and model-based reasoning and its application to physical system faults

Stamos T. Karamouzis
College of William & Mary - Arts & Sciences

An integration of case-based and model-based reasoning and its application to physical system faults

Karamouzis, Stamos T., Ph.D.

The College of William and Mary, 1993

An Integration of Case-Based and Model-Based Reasoning

and

its Application to Physical System Faults

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the degree of

Doctor of Philosophy

---

by

Stamos T. Karamouzis

1993

APPROVAL SHEET

This Dissertation is submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

[illegible]

Stamos T. Karamouzis

Approved, July 1993

[illegible]

Dr. William L. Bynum

Alan Pope

Dr. Alan Pope
NASA Langley Research Center

[illegible]

Dr. W. Robert Collins

[illegible]

Dr. Richard H. Prosl

[illegible]

Dr. Stefan Feyock
(Chair)

[illegible]

Paul C. Schutte
NASA Langley Research center

Dedication

This thesis is dedicated to my parents, Triantafillos and Vassiliki Karamouzi, and to my sister Yiota Karamouzi for their love, support, and sacrifices.

Table of Contents

Acknowledgments

The task of acknowledging all the people who contributed in this research is a very difficult one, since there so many people who contributed, influenced, encouraged, and supported this effort.

I particularly want to thank the following people:

My advisor, Stefan Feyock, for teaching me about artificial intelligence. Without his scientific knowledge, collegial advice, and unique sense of humor this thesis would not have been possible.

My committee members: Bill Bynum, Bob Collins, Alan Pope, Richard Prosl, and Paul Schutte.

My friends at William and Mary: Antonis, Ashok, Chris, Matt, Raja, Tracy, and others.

I want to especially thank Courtney, Cheavn, and Cameron Frantz for their support, concern and their cookies-and-cream ice-cream.

This work was supported by NASA grant NCC-1-159 and conducted at NASA/Langley Research Center in Virginia. I'm grateful to all members of the Human Automation Branch for providing me the resources and support necessary to carry on this research. Special thanks to Kathy Abbott, for giving the green light to every request; Paul Schutte, for carefully keeping an eye on the research; Steve Smith, for keeping the systems running; and John Barry for saving me every time the Lisp code became the monster in my bedtime nightmares.

List of Figures

Abstract

Case-Based Reasoning (CBR) systems solve new problems by finding stored instances of problems similar to the current one, and by adapting previous solutions to fit the current problem, taking into consideration any differences between the current and previous situations. CBR has been proposed as a more robust and plausible model of expert reasoning than the better-known rule-based systems. Current CBR systems have been used in planning, engineering design, and memory organization. There has been minimal work, however, in the area of reasoning about physical systems. This type of reasoning is a difficult task, and every attempt to automate the process must overcome the problems of modeling normal behavior, diagnosing faults, and predicting future behavior.

CBR systems are quite difficult to compare and evaluate, because until now there has been no common mathematical framework in which the systems can be described. The only avenue available at present for comparison and evaluation of CBR systems requires an intellectual synthesis of the semantics of the implementations. Important constraints on the operation of a CBR system are often hidden in obscure programming tricks in the system's source code.

This thesis presents a hybrid methodology for reasoning about physical systems in operation. Our methodology is based on retrieval and adaptation of previously experienced problems similar to the problem at hand. In this methodology the ability of a CBR to reason about a physical system is significantly enhanced by the addition to the case-based reasoner of a model of the physical system. The model describes the physical system's structural, functional, and causal behavior.

Additionally, this thesis presents a mathematical formalization of the case-based reasoning paradigm and a formal specification of the interaction of the CBR component with the model-based component of a case-based system. To prove the feasibility and the merit of such methodology, a prototypical system for dealing with the faults of a physical system has being designed and implemented. Testing has shown that this hybrid methodology allows the generation of diagnoses and prognoses that are beyond the capabilities of current reasoning systems.

"ουτω πανι τι ερουσιν οι πολλοι ημας,

αλλ οτι (επει) ο επαιων."

— **Socrates**

An Integration of Case-Based and Model-Based Reasoning

and

its Application to Physical System Faults

Chapter 1

CBR and Physical Systems

1.1 The Problem

We consider a physical system as a set of components connected together in a manner to achieve a certain function. *Components* are the parts that the system consists of, and may themselves be composed of other components. For example, an engine is a component in an airplane and it is composed of other components such as a compressor, a combustor, a fan etc. Components which are composed of other components are called *subsystems*.

Reasoning about physical systems is a difficult process, and every attempt to automate this process must overcome many challenges. Among these are the tasks of generating explanations of normal behavior, fault diagnoses, explanations of the various manifestations of faults, prediction of future behavior, etc. The reasoning process becomes even more difficult when physical systems must remain in operation. During operation, a physical system is changing dynamically by modifying its set of components, the components' pattern of interconnections, and the system's behavior. See **Figure 1.1** on page 11.

Explaining normal behavior is the process of elaborating the function of each subsystem and how this function contributes to the overall operation of the system. Explaining the operation of an automobile, for example, would require knowledge of the function of the carburetor, operation of the fuel pump, movement of the wheels, etc., and how all these affect each other and contribute to the final operation of moving the automobile. There are several approaches to explaining the normal behavior of physical systems by means of a model of the system. These approaches include naive physics [Hayes 1979], qualitative physics [deKleer 1985; Forbus 1985; Kuipers 1985], bond graphs [Rosenberg & Karnopp 1983, Feyock 1991], causality models, and others, each of them achieving various degrees of success and various advantages over the others.

Fault diagnosis is the process of explaining why the behavior of a system deviates from the expected behavior. Such diagnoses are the answers to the questions "Why has my watch stopped?" and "Why were the lights flickering after yesterday's storm?" Fault examples include a broken spring, a dead battery, a leak in a fuel line, etc. The task of diagnosis presents particular challenges such as identifying the faulty component, taking into consideration fault propagation, and accounting for multiple faults.

A number of systems have been developed to deal with these problems. Such systems fall into two categories. *Associational* or *shallow-reasoning systems* are systems that do diagnosis based on predefined links between sets of symptoms and pre-existing explanations [Buchanan & Shortliffe 1984]. These systems are fast but inflexible, since their lack of deep domain knowledge makes them incapable of dealing with problems outside their preset rule bases. *First-principle* or *deep-reasoning systems* use causal reasoning to produce explanations for the set of symptoms [Davis 1984]. These systems are more flexible, but are slower, since they must derive each new diagnosis from the underlying model.

In maintenance diagnosis, i.e. diagnosis of physical systems not in operation, it is sufficient to identify the source of the problem (faulty component) in order to determine which component(s) need to be repaired. In domains where the system is in continuous operation, however, it is desirable that the system operators be aware of fault consequences in order to facilitate corrective actions. A pilot who observes abnormal behavior in the plane's sensor values needs to know not only what the fault is, but also how the fault will propagate and what its subsequent effects will be.

Automating the process of predicting the future behavior of physical systems is a difficult task because physical faults manifest themselves in various ways and it is difficult to enumerate all possible consequences. Current efforts to incorporate prognostication features in diagnostic systems that reason from physical system models succeed in predicting the expected course of events but are limited by the level of detail of their models [Feyock & Karamouzis 1991]. For example, a model-based reasoning system that has a model of an airplane's functional and physical connections among components may, after establishing that the fan in the left engine is the faulty component, predict that the fault will affect the operation of the compressor since there is a functional link between the two components. Such a system is incapable, however, of deducing that flying fragments from the faulty fan may penetrate the fuselage and damage the right engine. Humans, on the other hand, are good at making such predictions, since their reasoning is based not only on preexisting models of the world, but also on previous directly or vicariously experienced events which remind them of the current situation.

1.2 Approach

This thesis presents a novel approach to dealing with physical systems while operating. The methodology presented here involves the use of case-based techniques in conjunction with models that describe the physical system. Case-Based Reasoning (CBR) systems solve new problems by finding solved problems similar to the current problem, and by adapting solutions to the current problem, taking into consideration any differences between the current and previously solved situations. Because CBR systems associate features of a problem with a previously-derived solution to that problem, they are classified as associational-reasoning systems.

We show a case-based reasoning methodology for fault diagnosis and prognosis of physical systems in operation. This methodology employs a hybrid reasoning process based on a library of previous cases and a model of the physical system that is used as basis for the reasoning process. This arrangement provides the methodology with the flexibility and power of first-principle reasoners, coupled with the speed of associational systems. Although domain independent, this work is tested in the aircraft domain.

In contrast to other CBR research efforts, each case in this methodology is not only a set of previously observed symptoms, but also represents sequences of events over a certain time interval. Such temporal information is necessary when reasoning about operating physical systems, since the set of symptoms observed at a particular time may represent improvement or deterioration from a previous observation, or may reveal valuable fault propagation information. In a jet engine, for example, the fact that the fan rotational speed was observed to be abnormal prior to an abnormal observation of the compressor rotational speed is indicative that the faulty component is the fan and that the fault propagated to the compressor, rather than the reverse.

The model represents the reasoner's knowledge of causal relationships between states and observable symptoms, as well as deep domain knowledge such as functional and physical connections among the components of the physical system about which the reasoner must reason. This research alleviates the knowledge acquisition problem to which current model-based systems are subject by letting each case of the CBR reasoning mechanism contribute its causal explanation, gained from adapting previous incidents, to the formation and maintenance of the causality model. The model can therefore be considered as a general depository of knowledge accumulated through time. In return the model aids the matching and adaptation processes of the CBR reasoning mechanism.

1.3 Methodology

The described research integrates case-based and model-based reasoning techniques for dealing with physical system faults. In order to demonstrate the challenges and benefits of such work a prototypical system is being designed and implemented in the aircraft domain. The system contains a self-organizing memory, as defined by [Shank 1982], for storing previously encountered problems. Each case has been represented in a memory organization packet (MOP) as implemented in [Riesbeck & Schank 1989].

Each case represents an actual aircraft accident case and consists of a set of features that identify the particular accident, a set of observable symptoms, and a causal explanation that describes the relationship between various states and observable features. The set of identifying features includes information such as aircraft type, airline, flight number, date of the accident, etc. The set of symptoms includes information about abnormal observations from mechanical sensors or "human sensors" such as the value of the exhaust gas temperature, the value of engine pressure ratio, the sound of an explosion, or the smell of smoke in the passenger cabin. These symptoms are presented in groups, each group representing a particular time interval. These time intervals are of unknown and uneven length; it is their ordering that it is of importance.

Additionally, the system incorporates a model, called the *world knowledge model*, that consists of deep domain information such as the physical and functional dependencies between the components of the physical system, and causal information describing the transitions between various states of the physical system. Along with the causal information between two states, e.g. "inefficient air flow" and "slowing down of the engine," the model maintains a frequency count of the number of times that the system witnessed that inefficient air flow caused the engine to slow down. The physical and functional connections are represented using LIMAP, a matrix-based knowledge representation tool [Feyock & Karamouzis 1992], and include information of the type "the Fan is connected to sensor N1 via a functional link," "the Fan is physically connected to the compressor." LIMAP provides an excellent tool for queries such as:

- "Is there a connection between the combustor and the turbine?"
- "If there is a connection, what kind of connection is it?"
- "Give me all the paths by which the turbine can be reached form the compressor."

The causality knowledge of the world model includes information such as "fan-blade separation causes the rotational speed of the fan to fluctuate" and "the rotational speed of the fan causes the engine pressure ratio to fluctuate."

When the system experiences a new set of symptoms it searches its case library for the most similar case. Based on the observation that similar faults manifest themselves in similar ways only during the first moments of the fault occurrence [AAIB-AAR-4/90], the system developed takes advantage of the available temporal information in each case, and tries to establish similarity based on the observable symptoms during the first moments of the fault occurrence. The input cases do not have to match exactly any previous cases in memory.

If the system finds and retrieves a similar case, the causal explanation of the retrieved case is adapted to fit the current case, and is stored in the case library for future usage. The system is provided with a set of adaptation rules which, in addition to adapting the retrieved causal explanation to fit the current case, find possible gaps in the causal explanation and fill in the missing causalities. This causal explanation connects the symptoms to a justifying cause, and thus the system's causal reasoning ability produces a causal analysis of the new case, rather than simply a reference to a previous solution. The new causal analysis is not only be stored in the case library as part of the input case, but is used to augment and modify the causality knowledge of the world model. The causal analysis will consist of a sequence of pairs of the type "event A causes event B," "event B causes event C" and so on. Each of these pairs is stored in the causality section of the model. In the case that the model already knows about the causal relation between two events from a previously seen case, the system updates the frequency count between the two events. The world model is therefore created based on the previous behavior of the physical system, and is constantly updated based on the current behavior, either by augmenting its previous causal knowledge or "becoming more sure" about causal relations.

Constant consultation of the model gives the system its prognostication ability. For example, having achieved a match of the current situation with a previous case where the faulty component was a bad fuel controller, the system hypothesizes that the same fault is occurring. By referencing the world model it is able to predict that an engine flameout may occur, although that did not happen in the retrieved case, because the model may have recorded at least one previous instance where this happened. The operator is provided with a list of possible consequences of the fault along with a frequency count of each one. Figure 1.1 is a diagram of the various modules involved in the reasoning system along with their interactions.

1.4 Results

Empirical testing of the methodology has lead to the following conclusions:

- Combining a memory of past cases with models combines the efficiency of associational reasoning with the flexibility of model-based reasoning.
- The integration of CBR and models enhances the ability of the model-based component by the CBR component's capacity to contribute new links into the causality model. The adaptation rules of the CBR component not only adapt the retrieved causal explanation to

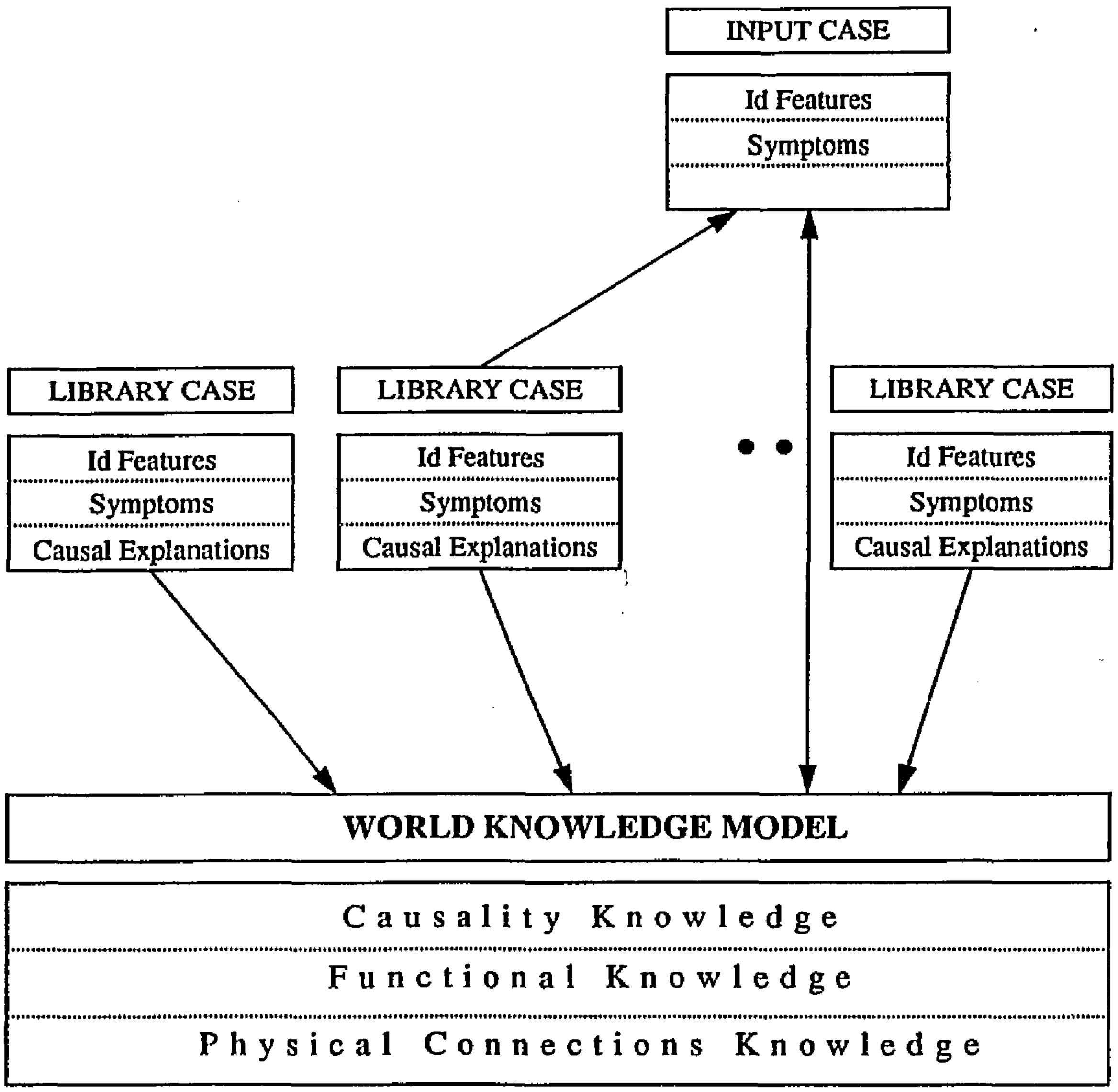

Figure 1.1: Models of the reasoning system and their interactions

- fit the current case, but they find possible gaps in the causal explanation and fill in the missing causalities. These additional causalities serve in the causal explanation of the current case and to expand the available knowledge to the model.

- The integration of CBR and models enhances the ability of the CBR component by using the model to aid the processes of matching, and adaptation. The model aids matching and adaptation in dealing with features which appear different on a superficial level, but are accounted for by the same initial cause.

- The use of the causality model provides enhanced fault-propagation forecast capabilities to the reasoner. The nature of the causality model (viewed as central depository) enables the reasoner to predict beyond the experiences of the retrieved case to the experiences accumulated by all previous cases.

1.5 Comparison to Other Work

Combined CBR and model-based reasoning (MBR) has been used primarily in engineering design. In the design domain a case consists of a design goal, a set of specifications for that goal, a set of constraints that must be met, and a plan for achieving the goal. CBR systems in this domain are faced with the challenge of using previous design plans in order to come up with a new design plan.

Recognizing the advantages of combining CBR and MBR, [Sycara & Navichandra 1989; Goel 1989; Goel & Chandrasekaran 1989] use device models in order to adapt old design cases. The fact that two design problems with different features might represent the same object if the features are studied based on their structural, functional, and causal behavior inspired them to use models to define the similarity between the two design problems.

Goel and Chandrasekaran [Goel and Chandrasekaran 1989] represent devices by a high-level model, called a *functional representation*, that describes the expected and unexpected behavior of the system. In contrast to our work their models are case specific and they don't use a causality model. Sycara and Navinchandra [Sycara and Navinchandra 1989] have proposed the use of causal models for adapting design cases in engineering domain. Apart from the differences in the task and the domain, their method differs from our work in that they use only causal models that contain no domain information on either the function nor the structure of the system. More importantly, Goel and Chandrasekaran along with Sycara and Navinchandra demonstrated how models may be used to aid case-based reasoning when dealing with devices that are not in operation. Dealing with devices that are in operation, as is done in our work, provides additional challenges since temporal information must be taken in to account. Our work explicitly represents and reasons about time when dealing with physical systems.

Although current CBR systems are goal-oriented and used mainly in planning, design, and memory organization, there is some work in the diagnostic domain. [Kolodner & Kolodner 1987] developed a diagnostic CBR system which reasons in the domain of medicine. Their system is more an application of dynamic memory as defined by [Schank 1982] than a diagnostic system. It organizes memory using Diagnostic MOP's and Process MOPs. Diagnostic MOPs are dynamic structures, updated from experience, that represent disease categories; Process MOPs are specialized structures that offer a predefined way to organize memory.

[Koton 1988] has combined model-based reasoning and CBR in medical diagnosis in a system called CASEY, which is based on a self-organizing memory for storing previously seen cases. Each case is comprised of a patient description and solution data. The patient description includes

signs and symptoms, test results, history, and current therapy information. The solution data includes a causal explanation of the symptoms, together with therapy recommendations. When the system is presented with a new patient description it attempts to retrieve a similar case and adapt the solution data of the retrieved case to fit the current patient description. If no acceptable previous case is found the system gives control to a model-based reasoning system called the Heart Failure Program. This program utilizes a network of causalities between various physiological states, and produces a causal explanation which describes the relationship between physiological states and observable features. Even when the CBR portion of CASEY is successful in producing a causal explanation of the observable features the user has the option of running the Heart Failure program. Although CASEY's CBR portion handles learning by storing newly created causal explanations in the case library, it has no provision of updating the causality model kept in the Heart Failure program. The model is therefore static, since it depends solely on a predefined causality network. In contrast to CASEY our work includes the provision of dynamically creating and maintaining the model from the set of previous behaviors of the physical system.

CASEY's algorithm includes the following stages [Koton 1989]:

- The phase of *retrieval* where CASEY retrieves from its case library a case similar to the new patient

- The phase of *justification* where CASEY evaluates the significance of any differences between the new case and the retrieved case using a set of principles for reasoning about evidence in causal explanations. These principles are used to: determine whether a feature in the retrieved case is ruled out by evidence in the input case; show that feature differences are insignificant or repairable; disregard differences in features that describe normal states, states from which no information is available or states that describe behavior with in the same qualitative region. If all differences between the new case and the retrieved

- case are judged insignificant or if the solution can be repaired to account for them, the match is said to be *justified.*

- The phase of *adaptation.* If none of the differences rule out the retrieved case, causal repair strategies are used to adapt the previous case's causal explanation to the new case. These causal repair strategies add or remove nodes and links to the transferred causal explanation. If all matches are ruled out, or if no similar previous case is found, CASEY uses the Heart Failure program to produce a solution.

- The *storage* phase where the new case and its solution are stored in the case library for use in future problem solving. Indexing is done using every feature that describes the case and does not discriminate significant or predictive features. In contrast, our work utilizes an indexing scheme which is based on assigning various weights on features that reflect the diagnostic importance of each feature.

[Hammond & Hurwitz 1988] report research in the domain of reasoning about physical systems. When given a case describing a fault together with its explanation, their system uses this explanation and a predefined causality structure of the domain to decide which features of the fault should be indexed. Their work does not include deep domain models that describe the structural and functional connections of the physical system, and targets the extraction of diagnostic features for storing cases, rather than performing complete reasoning about physical system faults. The importance of their contribution in the area of reasoning about physical systems lies in the development and use of a simple but powerful set of heuristics concerning causal relatedness in physical systems. These heuristics are used to evaluate the likelihood that two features are causally related in the event that the system's causal model is unaware of a causal chain between them. For example, dirt and grass covering a lawn mower may be predictive of a plugged air filter while a bent handlebar probably is not - although in neither case does there exist a direct causal chain from the failure to the feature. Their diagnostic algorithm is a variation of the one used for case-based planning [Hammond 86] and includes the following steps:

- The phase of *selection* where given an input case the observable features of the case are used to find similar cases in the case library. One of these cases is selected as the one that matches the best with the input case.

- The phase of *matching* where portions of the retrieved causal explanation are matched against features of the input case. By examining the status of the physical system further matches are done.

- The phase of *modification* where any deviations from the retrieved causal explanation are repaired using backward chaining and the causal relatedness heuristics.

- The phase of *connection* categorizes features into those that are explained by the normal use of the physical system, those that are explained by the causal chain leading to the failure, and those that remain unexplained. Features in the first category are connected into a model that describes the actions that are performed on and with the system, and features in the second category are connected in the causal explanation of the failure.

- *Extraction* is the phase where the features that are causally related to the failure form the list of candidate features for indexing. The unexplained features and the features that are explained by the normal use of the system are ignored.

- During the phase of *indexing* the input case is stored in the memory, indexed by the features that predict its applicability. These index features are comprised by those features

- that must be present in any instance of the diagnosed problem and those features that might be causally related, as determined by the relatedness heuristics.

[Turner 1988] reports research which presents an approach to diagnostic reasoning called *schema-based reasoning* (SBR), which allows a reasoner to access and use the most specific procedural information available for the problem at hand. By using schema-like information, the reasoner can bring specialized problem-solving procedures to bear on diagnostic problems. His ideas are demonstrated in MEDIC, an SBR diagnostic reasoner whose domain is pulmonology. MEDIC's memory is an interconnected set of discrimination nets, or hierarchies, in which the leaf nodes are cases and scenes, and the interior nodes are MOPs or schemata. Turner's work is more an application of memory organization than a diagnostic system. Because MEDIC does not allow cases of problem solving to be added to its memory in a manner implemented in every traditional CBR system, it is incapable of learning.

In contrast with our research, all of the CBR work mentioned in this section is reflected on specific applications with no foundations on any theoretical base. Formalizing the case-based reasoning paradigm is the major contribution of this work to the future CBR research efforts. Additionally, unlike other work, our research demonstrates the challenges of explicitly representing and reasoning about time. This is an important attribute in the diagnostic task since observed symptoms at a particular time may represent improvement or worsening due to the system's behavior at a previous time.

"Those who cannot remember the past

are condemned to repeat it."

— **Santayana**

"I have but one lamp by which my feet are guided,

and that is the lamp of experience.

I know no way of judging of the future but by the past."

— **Patrick Henry**

Chapter 2

Case-Based Reasoning

2.1 Case-Based Reasoning Paradigm

The basic cycle of a CBR system is "input a problem, find a relevant old solution, adapt it." When a problem is input to a CBR system, an analysis, performed by the system, determines the features relevant to finding similar cases. These features are called *indices*. Relevance is usually determined not by the obvious features of the input problem, but by abstract relations between features, absence of features, and so on. The problem of determining what extra, non-obvious features are needed for a particular domain is called the *indexing problem.*

Usually the indexes retrieve a set of potentially relevant old cases. The next step is to *match* the previous cases against the input and reject cases that are different from the input and determine which of the retrieved cases is the most similar. This similarity of cases is determined by how well they match on each feature, and how important each feature is. For example, when the visitor in Athens is confronted with the situation of using a bus, a previous experience of using the subway in Athens, and an experience of using the bus in London may be retrieved as relevant cases. Following a careful evaluation of the important features in each case the visitor may consider that his bus experience in London is more closely related to the current situation. In the current situation, a location match is considered of lesser importance than the type of the desired means of transportation, therefore the London experience forms the best match.

After a best match is determined it must be *adapted* to fit the current situation. During the adaptation process it must be determined what is different between the input and the retrieved best match, and then the solution associated with the retrieved case must be modified to take into account those differences. The modified solution becomes the solution of the input situation. How much adaptation needs to be done depends on the nature of the differences. In our example, very little adaptation must be done in the process of recognizing an Athenian bus since its differences with a bus in London are minor, but more adaptation has to be done in the process of getting a ticket if the experience of using the subway in Athens was the best match in the situation of using a bus in Athens. The following sections investigate with more detail the various phases of the CBR paradigm along with the structures used for organizing the memory.

2.1.1 Memory Organization

In the early 80's [Schank 1982] developed knowledge structures for organizing memory called Memory Organization Packages (MOPs). These structures involve standard AI concepts, such as frames, abstraction, inheritance, and so on. MOPs are used to represent knowledge about classes of complicated events and contain a set of *norms* which represent the basic features of a MOP, such as: what events occur, what goals are accomplished, what actors are involved, and so on. For example, the following two MOPs describe an event between Tim and David, and the outcome of the event:

```
fight-event-mop                          fight-outcome-mop
    ACTION    stab-mop                       STATE     dead-mop
    ACTOR     tim-mop                        ACTOR     david-mop
    OBJECT    david-mop
    FREQ      several-times-mop
```

Similar knowledge structures for organizing memory, called *scripts*, were developed by [Schank & Abelson 1977]. Scripts differ from MOPs because they are not organized into interlinked networks as MOPs. Additionally, scripts are static knowledge structures, but MOPs are used to form dynamically changing knowledge bases, i.e., systems that learn new knowledge in the process of understanding and problem solving. During the same time that scripts were proposed, [Minsky 75] proposed *frames*, which are analogous structures and at that time were used in the domain of visual processing.

A MOP that refers to an instance rather than a category is called an *instance MOP.* MOPs are joined together with links. [Riesbeck & Schank 1989] classify links into the following categories:

a. A MOP may be joined to a more specific version of itself. The specific version is called a *specialization* and the more general MOP is called an *abstraction.* The link that joins a specification and an abstraction is called an *abstraction link.* A network of MOPs, going from very specific instances at the bottom to very abstract general knowledge at the top, is called an *abstraction hierarchy.* In an abstraction hierarchy the features of each MOP are inherited by the MOPs below it. For example, if we represent the process of "getting a Ph.D. in computer science" in a MOP, then this MOP can be linked via an abstraction link to a MOP that represents the process of "getting a doctoral degree."

b. MOPs that represent events have *scene links* to various sub-events. The network of MOPs linked together by scene links is called the *packaging hierarchy.* In our example, passing an oral examination could be a scene in the "getting a doctoral degree" MOP.

c. In some systems a MOP may be linked to those instances from which the MOP was originally derived, or to prototypical examples of the MOP. These links are called *exemplar links.*

d. A MOP may be linked to instances of the MOP that involved an expectation failure via *failure links*. In our example, the "getting a doctoral degree" MOP may be linked to a particular instance MOP that describes the unsuccessful effort of a certain student to get a doctoral degree because he performed poorly in the required course-work.

e. Links that join a MOP with its specializations are called *index links*. Each index link is labeled with an attribute-value pair. These pairs of attributes and values are not features for the MOP. When a MOP is indexed by such a pair, then the pair automatically becomes a feature of the MOP and every other MOP under this MOP inherits this feature. In our example, "area of study" is an attribute and "Computer Science" a possible value. The index link "area of study = computer science" would link the "getting a doctoral degree" MOP with the "getting a Ph.D. in computer science" MOP. The network of MOPs that is formed by the index links is called the *discrimination net*.

Not every Case-Based Reasoner makes use of all of these kinds of links. For example, a Case-Based Reasoner that needs to classify hardware based on their CPU type may use abstractions instead of index links. This can be done by creating a set of abstraction MOPs under the hardware-type MOP, where each abstraction has only one slot, namely the slot for CPU type. Then the reasoner can put each particular hardware piece under the appropriate abstraction. The use of abstractions in this manner would subdivide the memory in the same way that it would be subdivided if the reasoner was to use index links such as "CPU type = *<some_type>*."

Organizing memory in abstraction hierarchies is a key characteristic of CBR systems that leads to efficient retrieval of previous cases. Case-based Reasoning systems that use MOPs to hierarchically organize their memory constantly add new instances, new abstractions, or new indexes. New instances are added during normal use of MOP memory for solving problems. These instances record experiences in terms of MOPs.

Additional knowledge may be recorded by creating abstractions. A simple approach for creating abstractions is a method called *similarity-based generalization*, where the formation of abstractions is done when a number of cases are discovered to share a common set of features. These common features are used to create the features of the new abstracted MOPs, and the unshared features are used as indexes to the original MOPs. A potential problem with *similarity-based generalization* is that it may form spurious generalizations until the case library is sufficiently large.

An approach that avoids this problem is *Explanation-based Generalization* (EBG) [Mitchell 1986; DeJong and Mooney, 1986]. In this method abstractions are made only when a plausible reason for their existence can be inferred, based on prior causal knowledge. The problem with systems that employ EBG is that they can end up doing a lot of work to create an abstraction for one-time only event. [Simpson 1985; Sycara 1987; Hammond 1988] employ a form of EBG called *failure-driven learning*, where in addition to the solutions the reasoner saves general explanations of why some solutions don't work.

2.1.2 Indexing

Retrieving relevant cases from memory can be a massive search problem. In order to make the retrieval process more selective and reduce the effect of memory size cases must be indexed by appropriate features. Throughout the literature there are several approaches involving the selection of an appropriate set of indices. The easiest approach is to use as indices all the features that form the description of a case. [Lebowitz 1987] uses inductive learning to determine relevant features which in return become indices. [Mark & Barletta 1988] use explanation-based techniques to identify predictive features for each case so they can serve as indices.

Although there are several context-dependent methods for selecting indices, there is a need for more study of this process, especially on methods for generating new indices dynamically.

2.1.3 Retrieval

Retrieval of relevant cases is one of the most crucial issues in CBR. Because most of the complex, real-world domains involve thousands of cases, the process of retrieving cases from memory becomes a massive search problem. The situation is complicated by the fact that we must perform some type of partial matching because an input case is unlikely to match exactly a previously stored case. Retrieval techniques depend on the structure of case memory, the information stored in each case, the features used as indices, the notions of similarity and relevance, and the available general knowledge about the domain.

To avoid exhaustive search, CBR search methods depend on a memory being organized in abstraction hierarchies. The search starts at the most general MOP in the abstraction hierarchy and proceeds downward only when a match is achieved at an abstraction MOP. Instance cases are therefore retrieved only when their abstractions match. To illustrate how this search works we present the following example in the domain of computer hardware. Let us assume that each computer system can be described in terms of three components: the type of the CPU, the type of the CRT, and the type of the keyboard.

Figure 2.1 on page 26 presents a snapshot of the reasoner's memory organization and we assume that the reasoner contains domain specific knowledge such as the fact that a 286 CPU is a CPU in the Intel family, a CPU in the Intel family is a CISC CPU, a CISC CPU is a single CPU, a CGA CRT is a color CRT, a color CRT is a CRT, a standard keyboard is a keyboard etc. We assume that an new computer comes and the task of the reasoner is to find the computers that are similar. The first component in the new computer is a 286 CPU, the second component a CGA CRT, and the third a standard keyboard. Note that the new computer is been represented in a MOP that contains three features, one for each component. Then the search proceeds in the following way:

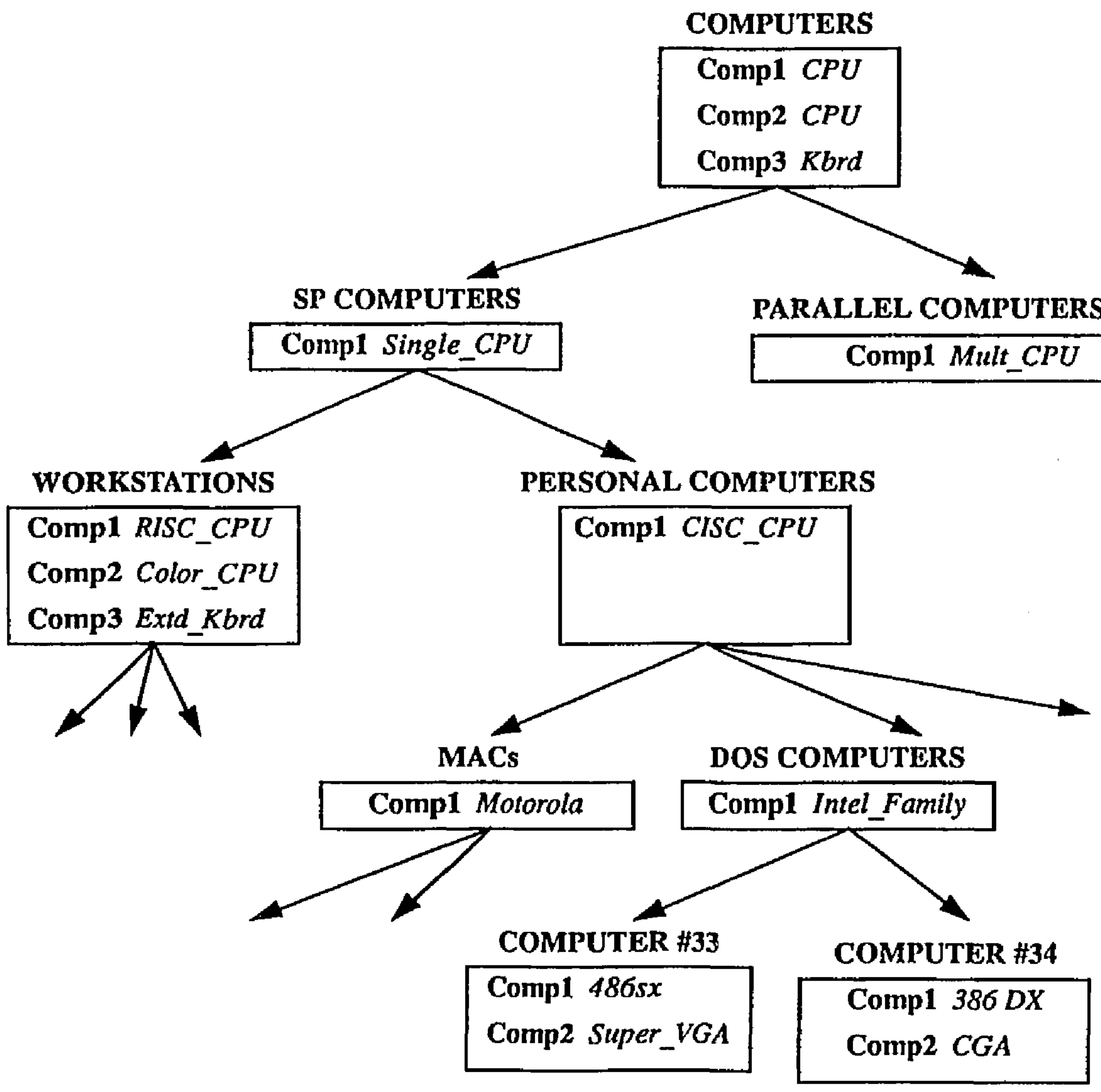

Figure 2.1: Snapshot of Case-Based Reasoner's memory orgainzation

Step 1: First level (top level) comparison. Each component that describes the new computer is been compared with the corresponding component that describes computers in general. Since a

286 CPU is a kind of CPU then we say that the value of the feature component1 in the MOP that describes the new computer *satisfies* the constrain imposed by the feature component1 in the MOP that describe computers in general. This constrain is the fact that component1 must be a CPU. Similarly because a CGA CRT is a CRT, and a standard keyboard is a keyboard then we say that the MOP New_Computer satisfies the MOP Computer and we move to the second step.

Step 2: Second level comparison. Each component that describes the new computer is been compared with the corresponding component that describes single processing computers (SP_Computers MOP). A 286 CPU is a single CPU thus the feature component1 in the New_Computer MOP satisfies the feature component1 in the SP_Computers MOP. The latter contains no component1, and component2 features thus these feature are inherited from the Computers MOP. Both of these feature are also satisfied by the corresponding features in the New_Computer MOP thus we move to the lower level under the SP_Computers MOP.

Step 3: Third level comparison. A CGA CRT is a Color CRT but a 286 CPU is not a RISC CPU thus the New_Computer MOP does not satisfy the constraints imposed by the Workstations MOP. Nest the New_Computer MOP is been compared with the Personal_Computers MOP. The latter inherits the values for component1, and component2 features from the SP_Computers MOP. A 286 CPU is a CISC CPU thus the Personal_Computers MOP is satisfied and the search continues with its children.

Step 4: Fourth level comparison. The New_Computers MOP can not satisfy the constraints of the MACs MOP but it does satisfy the DOS_Computers MOP thus the search continues with the children of that MOP.

Step 5: Retrieval. Since the children of the DOS_Computers MOP are instances of particular computers then these are retrieved as the computers that are similar to the new computer.

Earlier retrieval implementations [Kolodner 1983; Lebowitz 1983] use of redundant discrimination networks in order to guide the search, but later implementations [Kolodner 1988] used memories with distributed representations where cases were stored in pieces.

Along with the issue of reaching relevant cases the reasoner must face the problem of choosing one of the retrieved cases, the one that matches "best" the input case. The chosen case, called *most-on-point*, should be the one that addresses the reasoner's current problem in the best way. There are several approaches to this problem. The simplest tactic would be to accumulate a (weighted) count of the number of matching features between each retrieved case and the input case. While this may work in some domains, it is inappropriate for most domains since the importance of each feature is context dependent. [Kolodner 88] employs a method based on preference heuristics. [Rissland & Ashley 88] use the method of *dimensional analysis*. In the domain of legal reasoning they have developed special knowledge structures called *dimensions* which identify a factual feature that links operative facts to known legal approaches to those facts, specify which are the most important for this approach, and specify how a legal positions strength or weakness can be compared to other cases. [Stanfill 87] uses dynamically changing weighted evaluation functions. In all of these methods the common aspect is that all of the retrieved relevant cases are taken in consideration in choosing what is important for choosing the most-on-point case.

2.1.4 Adaptation

After the retriever finds the best match that it can in memory, the system proceeds to adapt the solution stored in the retrieved case to the need of the current situation. The adaptation process looks for salient differences between the retrieved case and the input and then applies rules that take those differences into account. Those adaptation rules can be much simpler than those required by a purely rule-based system. In a planning domain the adaptation rules note preconditions to steps that need to be met and suggest plans to achieve these preconditions. In a diagnostic task, the adaptation rules find gaps in an causal explanation and fill in the missing causalities.

A CBR system can get by with a much weaker set of adaptation rules, if the case library is broad enough. The process by which most people learned to find logarithms in high school demonstrates how a bigger case library can allow the use of significantly weaker adaptation rules and still get strong results. In the process of finding logarithms, the table of logarithms is analogous to the case library. Looking up the closest numbers is case retrieval and interpolating the answers using ratios is the adaptation rule. This simple rule yields reasonable answers only if the table has two numbers close to our number.

Types of Adaptation

[Riesbeck & Schank 89] describe two types of adaptation. *Structural adaptation* is the process of applying the adaptation rules directly to the solution stored in the retrieved case. [Hammond 88] uses it in the domain of cooking to modify previous recipes in order to come up with a new recipe, and [Bain 86] in the domain of legal reasoning in order to modify prior criminal cases.

The second type is *derivational adaptation*, where the rules that generated the solution in the retrieved case are re-run to generate the solution in the input case. Systems that use derivational adaptation store not only a solution with each case, but the planning sequence that constructed that solution [Simpson 1985]. An advantage of derivational adaptation is that requires fewer *ad hoc* rules [Hammond 1989]. Additionally, it can be used to adapt problem solving knowledge from other domains, rather than being restricted to within-domain solutions [Simpson 1985].

Using a particular type of adaptation does not imply exclusion of the other type. In reality CBR systems should have both structural adaptation rules to fix the "non-analyzed" solution, and derivational mechanisms to fix cases that are well understood by the systems. For example, solutions generated by the system itself are ideal for derivational adaptation.

Adaptation Techniques

The simplest adaptation technique is to do nothing and simply apply the solution of the retrieved case to the new situation. This is called *null adaptation* and comes up in tasks where, even though the reasoning to a solution may be complex, the solution itself is very simple. For example, when evaluating loan applications many factors must be considered, but the final answer is either accept or reject. Considering the fact that the real solution stored in each case is the chain of reasoning leading to a particular answer, the disadvantage of null adaptation is that does not provide to the user information such as how a particular answer was derived, what other answers are possible, and so on.

Parameterized solutions is another technique where given an input situation and the retrieved case, the retrieved and new problem descriptions are compared along the specified parameters. The solutions are then used to modify the solution parameters in the appropriate directions [Rissland and Ashley 1986; Bain 1986; Sycara 1987; Hammond 1989]. This technique is of value in modifying an existing solution, not creating a solution from scratch. It is a simple and powerful way to augment a case library, but is not a replacement for a good set of cases.

Abstraction and respecialization is a structural adaptation technique where, if a piece of the retrieved solution does not apply to the problem at hand, the system looks for abstractions of that piece of the solution that do not have the same difficulty. Then it tries to apply other specializations of the abstraction to the current situation [Alterman 1986; Kass 1986; Sycara 1987].

[Sussman 1975] proposed the notion of *critics* as a debugging tool for nearly correct solutions. His proposal was implemented in [Simmons 1988]. A *critic* looks for some combination of features that can cause a problem in a plan. Associated with different problems are strategies for repair. The feature combinations that are worth checking depends on how the plans are derived. In Sussman's work, a plan for achieving several goals simultaneously is derived by putting together plans that could achieve each goal independently. The critics then check if any plans interfered with each other, or if any plans are redundant. Critics as used in CBR systems can make only local changes to solutions, rather than globally reorganizing everything [Sycara 1987; Hammond 1989].

Reinstantiation is a derivational adaptational technique which operates not on the solution of the retrieved case, but on the method that was used to generate that solution [Simpson 1985; Hammond 1989]. Reinstantiation means replacing a step in a solution by taking the plan that generated that step and rerunning it in the context of the current situation. Since reinstantiating a plan is planning, the power of this technique is limited by the planning power of the reasoner.

2.1.5 Testing

As soon as the adapted solution becomes the solution of the input case, most CBR systems pass the solution through a tester. This phase is important in domains such as planning or legal reasoning where there is no unique "right" answer. One way to test the new solutions is by proposing hypothetical and counterexamples to test the robustness of the solution. Another way is to use the new solution as a probe into memory and try to find similar instances in the case library that lead to failure. Both methods are case-based, since they have to go into the case library and try to retrieve some other case. [Hammond 88] in the domain of planning employs simulation for testing. The idea is to pass the solution through a simulator and check the results of the simulation against the results from the CBR system.

2.1.6 Failure Explanation

Reasoning systems may fail at the testing phase when generating plans or do diagnoses, either because goals specified in the input are not achieved, or because implicit goals, not specified in the input, are violated. When a CBR reasoner fails, it has to explain its failure and repair it. In planning explanation comes before repair, and the repair is based on the explanation. In diagnosis, the repair has to come first.

The task of explanation is to generate a domain specific explanation of why the proposed solution failed. In a planning domain the explanation is a causal chain leading from the steps in the plan to the violation of the goal [Hammond 1988]. In a fault diagnosis domain, the explanation is a causal chain leading from the failure of some component, other than the one diagnosed as faulty, to the observed fault symptoms.

2.1.7 Repair

Given a solution, a failure report, and possibly an explanation, the task of repair is modification of the solution to remove the failure. In domains where explanation precedes repair, the explanation of the failure will usually provide clues to the repairs needed. In other domains, such as fault diagnosis, the only information available to the repair process is the diagnostic failure, i.e., "the component selected as cause of failure is functional." In these domains one repair strategy is to add whatever new information is available in the failure report and then search the case library for another best match. If the additional information causes a different case to be retrieved, then it should be adapted. If the same case is retrieved as before, then an alternative repair strategy is to try adapting the second-best match.

Whenever solutions fail and are repaired it is important to link the solution that didn't work with the one that finally did. This link will be useful when the same case fails to apply again, in some other situation. When this happens, the system can look at any other failures associated with this case and try to generalize what is common using either similarity-based or explanation-based generalization techniques. The goal is to find some characterization of the failing situations in order to avoid that class of failures in the future. For example, [Hammond 86] in the cooking domain employs a problem anticipation mechanism where the system (recipe planner), by noticing features in the input case that have previously contributed in past planning problems, anticipates planning problems in the current case. The fact that cooking beef and broccoli together makes the broccoli soggy, i.e., fails to achieve the goal of having a crisp vegetable, is worth remembering. Generalizing the failure into "cooking meat with a crisp vegetable makes the vegetable soggy" avoids subsequent failure when the system is asked to produce a recipe with chicken and snowpeas.

2.2 CBR versus Rule-Based Systems

CBR systems are an alternative to traditional Rule-Based (RB) systems. RB systems consist of a rule base of domain-specific knowledge, and a domain-independent rule interpreter that combines the rules to construct answers to problems [Buchanan & Shortliffe 1984]. RB systems are divided into *production systems* which contain rules of the form "IF some conditions are met THEN take some action", and *deductive systems* which contain rules of the form "IF some predicates are true, THEN conclude some other predicates are also true."

RB systems are flexible and can produce nearly optimal solutions, but are slow and prone to errors. Ease of adding a new rule or modifying an existing one is the major advantage of RB systems. RB systems are intuitive and better represent some kinds of knowledge that people seem to have. Rules capture "what to do knowledge" but not *deep domain knowledge* such as "why it works" or "what it means" [Chandrasekaran & Mittal 1982]. Another problem with RB systems is that the knowledge of the system is scattered among individual pieces. Therefore the more facts the system knows the slower it becomes. A third problem is that rules are not good structures for representing events.

In contrast to RB systems, CBR systems are restricted to variations on known situations, and produce approximate solutions. In realistically complex domains are quick and their solutions are grounded in actual experience. Most importantly cases support knowledge transfer of expertise and explanation better than rules do. Because human expertise is more like a library of past experiences than like a set of rules, using CBR systems makes the tasks of communicating expertise from domain experts to the system and justifying a solution from the system to domain experts much easier. Additionally, many real-world domains are so complex that it is impossible or impractical to always specify the rules that involved. By means of cases we can always extract solutions, albeit approximate, to problems by retrieving a case that demonstrates some degree of similarity with the current problem.

2.3 Prototypical CBR Systems

Since the early 80's several reasoning systems have been constructed that can be considered to follow the CBR paradigm. The following sections give a brief overview of exemplar CBR systems that demonstrate much of the work in the area, and have influenced the development of future systems.

2.3.1 IPP

IPP's (*Integrated Partial Parser*) [Lebowitz 1980] domain is international terrorism, where it is able to read texts about terrorist activities, store its interpretations in memory, and make generalizations. IPP's interesting characteristics include its memory structure, its rules for forming abstractions, and its use of memory to guide parsing. IPP is the first attempt at a computer system that uses dynamic memory structures (MOPs). Generalizations are made based on the assumption that similarities between the story being read and stories previously stored in memory represent generalizations that describe the world. These generalizations are used as a basis for organizing events and to guide future story understanding.

2.3.2 CYRUS

Along with IPP, CYRUS [Kolodner 1984] is another MOP-based story understanding system that focuses on how memory is used to answer questions after understanding. CYRUS' domain is international politics. The system uses two databases, one for former Secretary of State Cyrus Vance and one for former Secretary of State Edmund Muskie. Following the basic cycle of reading a story, storing its interpretations, and making generalizations, CYRUS is capable of answering questions such as "When did Vance meet Begin last?", "Has Vance talked to Gromyko recently?". CYRUS' power is demonstrated in the process of answering questions such as "Did Vance met Mrs. Begin?". Instead of performing the impossible tasks of exhaustive memory search or indexing every episode in advance under every possible question it answers the question by answering subsequent questions such as "When would Vance meet the spouse of a diplomat?: At a state dinner", "When would he go to a state dinner with Begin?", and so on.

2.3.3 MEDIATOR

MEDIATOR [Simpson 1985] is the first CBR system in the domain of dispute resolution. Given a conflict of goals between several parties and a MOP-based case library, it creates a new instance of a MOP to obtain some plan for resolving a dispute. Employing derivational adaptation, the system modifies previously stored plans in order to satisfy the current dispute. For example, when called to resolve the dispute between Egypt and Israel, it retrieved from memory a plan to settle a dispute between two children over the use of an orange. The retrieved case comprised of a plan to give each child the part of the orange that she wanted: one wanted the peel while the other wanted the fruit. Adapting the retrieved solution MEDIATOR came up with a solution to give Israel military control, but to give Egypt political and economic control. In cases where the proposed dispute resolution fails to satisfy the involved parties, it employs a failure-driven learning mechanism by storing a record of the failure in order to predict and avoid such failures in the future.

2.3.4 SWALE

The SWALE [Schank 86; Kass 86] system is a MOP-based explainer with a library of patterns for explaining why animals and people die. This library includes patterns such as old age, being run over by a car, and so on. When SWALE is given a death case which can not be explained by any of the normal explanation patterns, it searches its library for situations where the death pattern was abnormal. It then uses abstraction and respecialization to adapt the abnormal pattern to the current situation. For example, when SWALE was asked to explain the death of a healthy race horse, it found in its memory a case of spouse killing spouse for life insurance, and reasoned that the healthy horse was killed by the owner for property insurance.

2.3.5 PLEXUS

Although PLEXUS' [Alterman 1986] memory organization is not based on MOPs and its case library is trivial, its adaptation mechanism is of interest. PLEXUS uses abstraction and respecialization to adapt previous plans for riding San Francisco's subway into a plan for riding New York's subway system. Initially PLEXUS uses null adaptation to adapt San Francisco's plan but when pieces of the plan fail then the system employs abstraction and respecialization. For example, San Francisco's plan calls for getting a ticket from a machine but in New York there no ticket machines. The system abstracts from the concept of "get ticket from machine" to "get ticket", then specializes to "get ticket from ticket booth," as in the plan for going to a theater.

2.3.6 JUDGE

JUDGE [Bain 1986] is a CBR system in the domain of "common-sense ethical reasoning" for criminal sentencing. The input is a description of a criminal case, along with the charge, the events that occurred, and the legal status regarding crimes of this nature. The case library contains previous crimes and the sentences determined for each. During its first stage of operation, JUDGE interprets the input case by inferring the seriousness of the crime, the motives of the actors in the current case, and determining the extent to which each offender was justified in acting violently, with the help of interpretations assigned to previous cases. It follows a retrieval phase along with structural adaptation of previous sentences in order to ensure that differences in sentence severity between crimes corresponds to differences in heinousness of the crimes. At the end a generalization phase forms sentencing rules when it finds it has several similar cases with similar sentences.

2.3.7 MBRtalk

MBRtalk [Stanfill & Waltz 1986] performs a word pronunciation task. By using a case library of several thousand words along with their pronunciations, it achieves 88% predictive accuracy in its task of mapping letters to phonemes. MBRtalk is a memory-based system rather than a traditional CBR system. In contrast with CBR systems which employ search methods which depend on a memory being organized in abstraction hierarchies, MBRtalk deals with the entire memory. It relies on parallel architectures with enough processors to facilitate simultaneous search for a partial match between the input word and every case in the memory. Selection is done by retrieving only the words that achieve some degree of partial match after the application of an evaluation function to each word in the case library. A crucial issue in this reasoning scheme is the choice of the appropriate evaluation function that reflects the case selection.

2.3.8 CHEF

CHEF [Hammond 1988] is a case-based planner that builds new recipes out of its memory of old recipes. CHEF's input is a set of goals for different tastes, textures, ingredients and types of dishes and its output is a plan, a single recipe, that satisfies all of the users goals. Much of CHEF's planning power lies in its ability to predict and avoid failures it has encountered before. The following are the basic stages of a case-based planner such as CHEF. *Problem anticipation* is the stage in which the planner, by noticing features in the input case that have previously contributed in past planning problems, anticipates planning problems in the current case. During the stage of *plan retrieval* the most similar case with the input case is retrieved from memory. *Plan modification* is the stage where the plans of the retrieved case are adapted to satisfy the goals of the new case. *Plan repair* is the stage where in case of a plan failure the planner finds different strategies for repair by building a causal explanation of the failure. During the stage of *credit assignment* the planner uses a causal explanation of why a failure occurred in order to identify the features of the input case that led to the failure, and mark them as predictive features. At the final stage of *plan storage* plans are placed in the case memory, indexed by the goals that they satisfy and the problems that they avoid.

Chapter 3

Formal Specifications

3.1 Rationale

In order to obtain a precise picture of the memory organization and various phases of the reasoning paradigm presented in this thesis we have developed a formal specification of the memory organization and various phases of this reasoning paradigm.

3.2 Models

The methodology presented in this thesis requires the availability and use of the following models: a functional dependency model, a physical dependency model, a causality model, and a manifestation model.

Let K, U, E, Y be finite sets of abstract symbols where $E \subseteq U$ and $Y \subseteq U$. We give the following interpretation to these sets.

- K as the set of components that comprise a physical system.
- U as a set of phenomena, events, occurrences, or symptoms that can occur during the operation of a physical system.
- E as a set of various events
- Y as a the set of symptoms

Components of a physical system may operate in two states: *normal*, and *abnormal*. Intuitively, a component k is said to be in an abnormal state if the operational behavior of the component deviates from the expected one. [We do not describe how component states are classified as normal or abnormal. We merely require that they be labelled one or the other.]

Definition 3.1: A *dependency model* (*d-model*) D is a relation on K, that is, $D \subseteq (K \times K)$.

Given a d-model D on K and components $k_1, k_2 \in K$, we say k_2 *depends on* or *is dependent on* k_1 if $(k_1, k_2) \in D$. [Note the reversal of indices.]

We view a d-model as a directed graph with nodes connected by arrows. Each node is a member of K and each edge represents a dependency (physical or functional) between the members of K. Intuitively a component k_2 is *physically* dependent on component k_1 if damage to k_1 can propagate through space to k_2. A component k_2 is *functionally* dependent on component k_1 if the operation of k_2 depends on the operation of k_1. [We do not describe how dependencies are classified as physical or functional. We merely require that they be labelled one or the other.]

Given a dependency, if the damage propagates instantaneously then the dependency is called an *immediate dependence*. If the damage requires an arbitrarily long time period to propagate, then the dependency is called a *non-immediate dependence*. [Again, we do not describe how dependencies are classified as immediate or non-immediate. We merely require that they be labelled one or the other.]

Definition 3.2: A *functional dependency model* F is a d-model containing only functional dependencies.

F can be partitioned into two subsets F_i and F_n i.e. $F = F_i \cup F_n$ and $F_i \cap F_n = \emptyset$. F_i and F_n represent functional dependencies that are immediate and non-immediate respectively.

Definition 3.3: A *physical dependency model* P is a d-model containing only physical dependencies.

P can be partitioned into two subsets P_i and P_n i.e. $P = P_i \cup P_n$ and $P_i \cap P_n = \emptyset$. P_i and P_n represent physical dependencies that are immediate and non-immediate respectively.

Figure 3.1 shows the classification of dependency models:

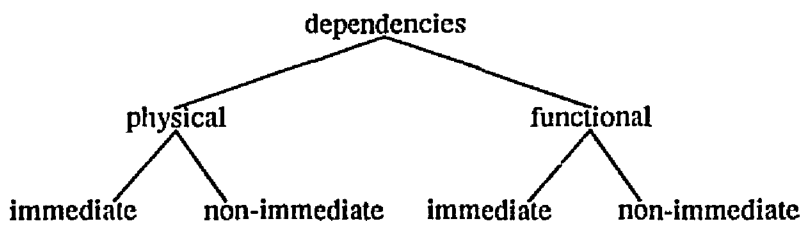

Figure 3.1: Classification of dependencies

Definition 3.4: A *causal model* Z is a relation on E, that is, $Z \subseteq (E \times E)$.

Given a causal model Z and events $e_1, e_2 \in E$, we say that e_1 *causes* e_2 if $(e_1, e_2) \in Z$.

Definition 3.5: A *manifestation model* Φ is a relation from K to U, that is, $\Phi \subseteq (K \times U)$.

Given a manifestation model Φ, a component $k \in K$, and a phenomenon, event, occurrence, or symptom $u \in U$, we say that the component k has been observed to *manifest* itself *as* u if $(k, u) \in \Phi$.

This thesis presents a hybrid methodology for reasoning about physical systems in operation. This methodology is based on retrieval and adaptation of previously experienced problems similar to the problem at hand. In this methodology the ability of the reasoner to reason about a physical system is significantly enhanced by the addition and utilization of the four models defined in this section. Section 3.3 presents a formalization of the Case-Based Reasoning paradigm and Section 3.4 shows how the models are utilized in Case-Based Diagnosis.

3.3 Case-Based Diagnosis

The structures used by the majority of CBR researchers are Memory Organization Packets (MOPs) as defined in [Schank 1982]. MOPs are frame-like structures that consist of attribute/value tuples called *slots*. The value of a particular slot may be another MOP, etc. With MOPs the memory is partitioned in a hierarchical way so that MOPs are abstractions or specializations of other MOPs.

Definition 3.6: A case memory CM is a system $(M, R, A, \Omega, ?, \sigma, \mu_{root})$ satisfying:

- M is a finite set of *memory organization packets* (MOPs),

  where $K, U, E, Y \subseteq M$
- R is a finite set of *slot roles*, satisfying $M \cap R = \varnothing$,
- A is a transitive, reflexive, antisymmetric relation on M, that is, $A \subseteq (M \times M)$. Furthermore, A must satisfy the ***abstraction constraint*** for certain of the MOPs in M. [See Definition 3.12.]
- Ω, the ***null element***, is an element where $\Omega \notin M$
- $?$, the ***unsolved element***, is an element where $? \notin M$
- σ is a function $\sigma: M \rightarrow 2^{(R \times M')}$, where $M' = M \cup \{\Omega , ?\}$, that satisfies the con-

straint: $\forall \mu \in M$, if $(\rho, \mu_1), (\rho, \mu_2) \in \sigma(\mu)$, then $\mu_1 = \mu_2$.

- $\mu_{root} \in M$ satisfies: $\forall \mu \in M$, $(\mu, \mu_{root}) \in A$.

Note A is a *partial order* on M, and the partially ordered set (M, A) has μ_{root} as its maximal element. A is called the abstraction relation, and the interpretation of $(\nu, \mu) \in A$ is that μ is more abstract than ν.

Definition 3.7: Given a case memory CM = $(M, R, A, \Omega, ?, \sigma, \mu_{root})$ we define the function

$$a: M \rightarrow 2^M$$

by $a(\nu) = \{\mu \mid (\nu, \mu) \in A$, and if $(\nu, \pi), (\pi, \mu) \in A$ then $\nu = \pi$ or $\pi = \mu\}$

Given the MOP $\mu \in M$, we call members of $a(\mu)$ the *abstraction* MOPs of μ. The elements of $a(\mu)$ are the minimal MOPs among all MOPs that are more abstract than μ.

Definition 3.8: Given a case memory CM = $(M, R, A, \Omega, ?, \sigma, \mu_{root})$ we define the function

$$s: M \rightarrow 2^M$$

by $\mu \in s(\nu) \Leftrightarrow \nu \in a(\mu)$

Given the MOP $\mu \in M$, we call the members of $s(\mu)$ the *specialization* MOPs of μ.

If we view a as a multi-valued function $a: M \rightarrow M$ where $a(\mu)$ is the set of values a assigns to μ, then s can be viewed as the (multi-valued) inverse of a.

Definition 3.9: Given a case memory CM = $(M, R, A, \Omega, ?, \sigma, \mu_{root})$, a *slot* is an element of $R \times M'$ $= R \times (M \cup \{\Omega, ?\})$.

Given a slot $\lambda = (\rho, \mu)$, ρ is called the *role* of λ, and μ is called the *filler* of λ. If $\lambda \in \sigma(\nu)$, then we denote μ as $\nu.\rho$. The constraint on σ makes the notation $\nu.\rho$ well-defined. If $\mu = ?$ then ρ is called a *goal*. Given $\nu \in M$, we denote the *set of roles associated with* ν

$$\{\rho \in R \mid \exists\, \mu \in M' \text{ so that } (\rho, \mu) \in \sigma(\nu)\}$$

as $\nu.R$.

Definition 3.10: $\mu \in M$ is a *slotless* MOP if $\sigma(\mu) = \varnothing$.

Definition 3.11: A MOP μ_i is an *input case* if

- $\sigma(\mu_i)$ contains at least one goal,
- $a(\mu_i) = \varnothing$, and
- $s(\mu_i) = \varnothing$.

All other MOPs are called library cases.

For notational convenience, we partition M into two disjoint sets M_i and M_a. M_i is the set

$$\{\mu \in M \mid s(\mu) = \varnothing\}$$

of *instance* MOPs and $M_a = M - M_i$ is the set of *abstraction* MOPs. Instance MOPs have no further specialization MOPs. Abstraction MOPs are the abstractions of other MOPs.

We are now in a position to define the *abstraction constraint* on A (or, equivalently, on a). There are limitations concerning which MOPs may be members of the abstraction set $a(\mu)$ of an abstraction MOP μ. Abstraction constraints may be specified in various ways. A particularly simple one would be to require that every member of $a(\mu)$ contain the same set $\mu.R = \{\rho \in R \mid (\rho, f) \in \sigma(\mu)\}$ of slot roles as μ, and that the fillers of corresponding roles be identical. That would be a strict constraint since it requires an exact match. We use a more relaxed abstraction constraint by requiring that the fillers of some members of $a(\mu)$ be abstractions of the corresponding fillers in μ.

We found that the abstraction constraint used in CBR system implementations reported by [Schank 1982; Riesbeck & Schank 1989] works for our purposes.

Definition 3.12: The *abstraction constraint* specifies that a MOP μ can be an abstraction of a MOP μ', i.e. $(\mu', \mu) \in A$, if and only if

1. μ is not an instance MOP, and
2. μ is not a slotless MOP, and
3. $\forall \rho \in \mu.R$, if $\rho \in \mu'.R$, then $\mu'.\rho$ must satisfy $\mu.\rho$.

A filler $f' \in M'$ is said to *satisfy* the conditions specified by another filler $f \in M'$, when one or more of the following conditions is true:

- f is Ω
- f is an abstraction of f', that is, $(f', f) \in A$.
- f is an instance MOP and f' is Ω
- f is not slotless, f' is not Ω, $f.R \subseteq f'.R$, and $\forall \rho \in f.$R, and $f'.\rho$ satisfies $f.\rho$.

We define the following operations on a CM:

Insertion of an instance MOP $\mu_i \in M_i$ into a case memory is the process of determining the set $a(\mu_i)$. (IFor abstraction MOPs $\mu \in M_a$, $a(\mu)$ is already specified by the user and the abstraction constraint.) The set $a(\mu_i)$ is determined in the following way:

$$a(\mu_i) = \{ \mu \in M_a \mid (\mu_i, \mu) \in A \text{ in accordance with the } \textit{abstraction constraint} \text{ and}$$
$$\neg \exists \mu' \ni : \mu' \in s(\mu) \text{ and } \mu_i \text{ satisfies } \mu' \}$$

In other words the MOP μ_i becomes a specialization of the most specialized MOPs in the CM whose abstraction constraint it satisfies.

Given the set $a(\mu_i)$ of an input case μ_i, *matching* is a mapping $\pi: M \rightarrow 2^M$ where the range of $\pi(\mu_i)$ is a set S is defined as follows:

$$S = \{ \mu \in M \mid \mu \in s(a(\mu_i)), \mu \neq \mu_i \}$$

S is called the set of *siblings* of μ_i, and consists of the MOPs having a parent in common with μ_i.

The input case is mapped into the member of S that best matches the input case based on some metric. Recall that a metric is a distance measure Δ satisfying the following four properties:

- $\Delta(\alpha, \beta) \geq 0$
- $\Delta(\alpha, \beta) = \Delta(\beta, \alpha)$
- $\Delta(\alpha, \alpha) = 0$
- $\Delta(\alpha, \beta) + \Delta(\beta, \gamma) \geq \Delta(\alpha, \gamma)$

Finding a generally suitable definition of Δ is one of the major current research problems in CBR. The simplest measure of dissimilarity between two cases is the number of slots for which they have different fillers. It is defined as follows:

$$\Delta(\mu_1, \mu_2) = \sum_{\rho = \rho_1}^{\rho_n} \delta(\mu_1.\rho, \mu_2.\rho)$$

where $\{ \rho_1, \rho_2, \ldots, \rho_n \} \subseteq \mu_1.R$ and

$$\delta(\mu_1.\rho, \mu_2.\rho) = \begin{cases} \text{if } \mu_1.\rho = \mu_2.\rho \text{ then } 0 \\ \text{otherwise } 1 \end{cases}$$

3.4 Use of Models in Case Based Diagnosis

Diagnosis is the process of replacing the filler *?* of unsolved slots by an appropriate member of M, in particular replacing the *?* fillers of the slots *fault* and *causal explanation* (abbreviated ce). This replacement is taking place during the *adaptation phase.* The filler of *fault* in some library case μ, i.e. μ.fault, is a MOP, whereas μ.ce is a MOP designating a set of tuples $X \subseteq (E \times E)$ such that for every $(e_1, e_2) \in X$ e_1 causes e_2.

The utilization of models in case-based diagnosis takes place in the adaptation phase. It is done as follows:

Let μ_i be an input case

$\pi(\mu_i) = \mu$

$d = \Delta\,(\mu_i, \mu)$

$\vartheta \geq 0$, is some threshold value

μ.ce is the causal explanation of μ

μ_i.ce is the causal explanation of μ_i. Initially μ_i.ce = *?*

Step 1:

μ_i.ce := μ.ce

$$\mu_i.\text{fault} := \begin{cases} \mu.\text{fault}_a & \text{if } d > \vartheta \\ \mu.\text{fault} & \text{if } d < \vartheta \end{cases}$$

where $\mu.\text{fault}_a$ is non-deterministically chosen member of $a(\mu.\text{fault})$

Step 2:

```
Case 1:   { $\mu_i$ and $\mu$ have identical symptoms }
          If, every $(\rho, f_i) \in \sigma(\mu)$ where $\rho \in Y$, the set of symptoms
          $\mu_i.\rho = \mu.\rho$
          then ; { i.e. adopt $\mu$.ce unchanged }

Case 2:   { $\mu_i$ has symptoms that do not appear in $\mu$ }
          If there is at least one $(\rho, f_i) \in \sigma(\mu_i)$
          where $\rho \in Y$,
          $\mu_i.\rho \neq$ 'normal'
          and $\mu.\rho =$ 'normal'
          then $\mu_i$.ce := $\mu$.ce

          Subcase 1:
              if $(e, \rho) \in Z$, then $\mu_i$.ce := $\cup\, \mu_i$.ce $(e, \rho)$

          Subcase 2:
              if $(k_1, k_2) \in F$,
              $(k_2, \rho) \in \Phi$,
              $(k_1, e) \in \Phi$,
              and $\mu_i.e \neq$ 'normal'
              then $\mu_i$.ce := $\mu_i$.ce $\cup\, (e, \rho)$
```

Case 3: { has symptoms that do not appear in μ_i }

If there is at least one $(\rho, f) \in \sigma(\mu)$

where $\rho \in Y$,

$\mu_i.\rho$ = 'normal',

$\mu.\rho \neq$ 'normal',

and $(\phi, \rho) \in \Phi$, { Φ is the manifestation model }

then,

Subcase 1:

if $(k_1, k_2) \in F_n$,

$(k_1, \phi) \in \Phi$,

and $(k_2, \rho) \in \Phi$,

then for every $(\phi, \rho) \in \mu_i.ce := \mu_i.ce - (\phi, \rho)$

Subcase 2:

if $(k_1, k_2) \in F_i$,

$(k_1, \phi) \in \Phi$,

and $(k_2, \rho) \in \Phi$,

then μ is rejected as the most similar case for μ_i.

In practice we retrieve the next closest (in terms of Δ) case from the set S.

Chapter 4

A Prototype

4.1 Introduction

The described research in this thesis integrates case-based and model-based reasoning techniques for dealing with physical system faults. In order to demonstrate the challenges and benefits of such work a prototypical system called Epaion has been designed and implemented in the aircraft domain.

Epaion contains a self-organizing memory structured as a frame-based abstraction hierarchy, as defined by [Schank 1982], for storing previously encountered problems. Currently each case has been represented in a memory organization packet (MOP) as implemented in [Riesbeck and Schank 1989].

Each case represents an actual aircraft accident report and consists of a set of features that identify the particular accident, a set of observable symptoms, and a causal explanation that describes the relationship between various system states and observable features. The set of identifying features includes information such as aircraft type, airline, flight number, date of the accident, etc. The set of symptoms includes information about abnormal observations from mechanical sensors such as the value of the exhaust gas temperature, the value of engine pressure ratio, or from "human sensors," such as the sound of an explosion or the smell of smoke in the passenger cabin.

In contrast to other CBR research efforts, each case in our methodology consists not only of a set of previously observed symptoms, but also represents sequences of events over certain time intervals. The time intervals are of unknown and uneven length; it is their ordering that it is of importance. Such temporal information is necessary when reasoning about operating physical systems, since the set of symptoms observed at a particular time may represent improvement or deterioration from a previous reading, or may reveal valuable fault propagation information. In a jet engine, for example, the fact that the fan rotational speed was observed to be abnormal prior to an abnormal observation of the compressor rotational speed is indicative that the faulty component is the fan and that the fault propagated to the compressor, rather than the reverse.

In addition, the system incorporates a model, called the *world knowledge model*, that represents the reasoner's knowledge of causal relationships between states and observable symptoms, as well as deep domain knowledge such as functional connections among the components of the physical system about which the reasoner must reason.

4.2 The domain

Epaion is being designed and implemented in the aircraft domain. Several aspects of the aircraft domain make automation of in-flight diagnosis challenging. In contrast with non-operative diagnosis (i.e., diagnosis of systems that can be shut down), symptoms in aircraft subsystems may change with time because of failure propagation. Information about the operational status of many aircraft components may be unavailable or incomplete due to limited instrumentation, and safety and comfort considerations place further constraints on in-flight testing.

Automation of in-flight fault diagnosis and prognosis can be used as an aid to the flight crew for early detection of a problem or failure. This provides the crew with more time to respond more effectively and reduce potential damage due to the failure.

The aircraft model used in this research is the same one used by [Abbott 1990] in her work on fault diagnosis. It is a simplified model of the propulsion system of a two-engine civil transport. This system consists of two turbofan engines and a fuel subsystem. A total of nine components are included. Four of them are sensors.

A turbofan engine was chosen since it is commonly used on civil transport aircraft. [Abbott 1990] describes the function of the engine as follows: The air enters the fan, a low-pressure compressor. The fan compresses the air, which flows to the high-pressure compressor. There the air is compressed further. It passes to the combustion section, which sprays fuel to mix with the highly compressed air, and ignites them. Ignition increases the velocity and temperature of the air, turning the turbines as the air flows to the exhaust section. The turbine section is divided into two stages. These two stages are connected to the fan and compressor with concentric shafts. The first turbine stage drives the compressor and the second stage drives the fan.

The engine has five sensors whose reading provide the following parameter values: N1, N2, Fuel flow (FF), exhaust gas temperature (EGT), and engine pressure ratio (EPR). The N1 and N2 sensors measure the rotational speeds of the fan and high-pressure compressor, respectively. The fan and compressor generally rotate at different speeds because they are connected to different turbine stages. Fuel flow measures the rate at which the fuel is entering the engine. EGT is the exhaust gas temperature. EPR is a ratio of the air pressure at the engine inlet. Figure 4.1 shows the schematic of a turbo-fan jet engine.

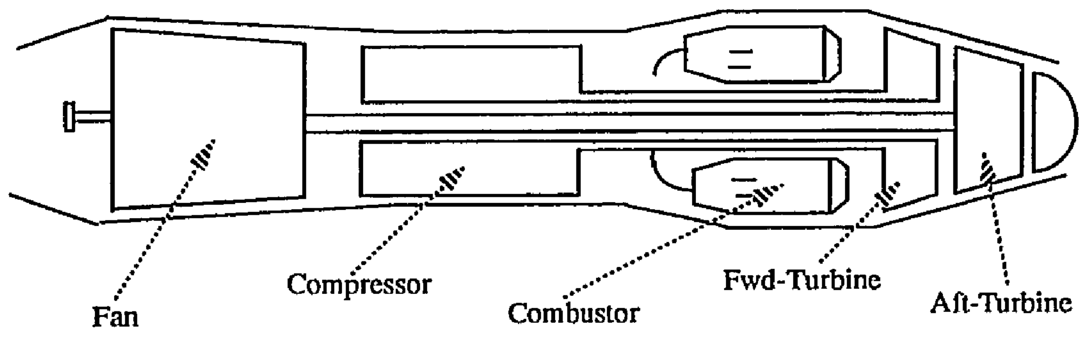

Figure 4.1: Schematic of a turbofan jet engine

4.3 Knowledge Sources

Epaion draws its power from several knowledge sources, including a library of aircraft accident/ incidents; a functional dependency model with deep domain information about the functional dependencies between the components of the aircraft; and a model representing causal information concerning transitions between various states of the aircraft.

4.3.1 Case Library

Epaion maintains a library of actual aircraft accident/incident scenarios called cases. Each case consists of a set of features that identify the particular scenario, a list of the relevant context variables and their particular status, a set of observable symptoms, the fault, and a causal explanation that connects the observable symptoms to a justifying cause. The set of identifying features includes information such as aircraft type, airline, flight number, date of the accident, and similar data. The list of context variables includes information such as the phase of flight, the weather, etc. The set of symptoms includes information about abnormal observations from mechanical sensors such as the value of the exhaust gas temperature, the value of engine pressure ratio, or from "human sensors," such as the sound of an explosion, or the smell of smoke in the passenger cabin. Cases containing all of this information are called library cases, whereas cases where the fault and the causal explanation are not available are called input cases.

In contrast to most other CBR research efforts, each case in our methodology consists not only of a set of previously observed symptoms, but also represents sequences of events over certain time intervals. The time intervals may have unknown and unequal lengths; it is the event ordering that it is of importance. Such temporal information is necessary when reasoning about operating physical systems, since the set of symptoms observed at a particular time may represent improvement or deterioration from a previous reading, or may reveal valuable fault propagation information. In a jet engine, for example, the fact that the fan rotational speed was observed to be abnormal prior to an abnormal observation of the compressor rotational speed is indicative that the faulty component is the fan and that the fault propagated to the compressor, rather than the reverse.

The following is an example of an actual case:

Identification Features:

Id: NTSB-AAR-76-19

Date: November 12, 1975

Airline: Overseas National Airways

Flight: Flight 32

Aircraft: DC-10-30

Context Variables

Phase of Flight: Take off

Symptoms

Fuel Flow: Initially normal, then fluctuating, then low

N1: Started fluctuating, then became high, then low

N2: Initially normal, then fluctuating, then low

EGT: Initially normal, then became high

EPR: Initially normal, then became high, then low

Fault

Bird ingestion

Causal Explanation

Bird ingestion caused fan blade damage,

which in return caused fan rotor imbalance,

which in return caused abnormal rotational speed of the fan.

Also the fan rotor imbalance caused abnormal rotational speed of the compressor.

The abnormal rotational speed of the compressor caused abnormal fuel flow,

it also cause abnormal exhaust gas temperature.

The abnormal fuel flow caused abnormal exhaust pressure ratio.

4.3.2 Causality Model

Epaion's causality model contains information such as ''fan-blade separation causes the rotational speed of the fan to fluctuate'' and ''the rotational speed of the fan causes the engine pressure ratio to fluctuate." Along with the causal information between two states, e.g. ''inefficient air flow" and ''slowing down of the engine", the model maintains a frequency count of the number of times that the system witnessed that inefficient air flow caused the engine to slow down.

Our research alleviates the knowledge acquisition problem to which current model-based systems are subject by letting each case of the CBR reasoning mechanism contribute its causal explanation, gained from adapting previous incidents, to the formation and maintenance of the causality model. This model can therefore be considered as a general depository of knowledge accumulated through time. In return the model aids the matching and adaptation processes of the CBR reasoning mechanism and enables Epaion to make prognoses that are beyond the knowledge of each individual library case.

4.3.3 Functional Dependency Model

The functional dependency submodel is a digraph model of an aircraft system, with nodes representing primitive components, and arrows connecting (linking) nodes representing functional dependencies. Component B is said to be *functionally dependent* on component A if the proper functioning of B depends on the proper functioning of A. For example, the control surfaces of an aircraft are functionally dependent on the hydraulic system, since they will cease operating if the latter fails. The functional dependency submodel contains two kind of arrows, representing immediate and non-immediate links between components. Two components C_1 and C_2 are connected via an immediate link (I-link) when abnormal function of C_1 at time t_1 results in abnormal function of c_2 at time t_2 and $t_1= t_2$. If $t_2 \geq t_1$ then C_1 is connected to C_2 via an non-immediate link (N-link). For example, the engine driven pump (EDP) bypass valve is connected via an N-link to the EDP filter, but the EDP filter is connected to EDP bypass valve via an I-link.

In order to efficiently represent the Functional and Physical Dependency Models a modeling tool named LIMAP was developed. This tool is oriented toward efficient information representation/ manipulation over fixed finite domains, and quantification over paths and predicates. The initial motivation for the creation of such a system was the fact that the need for such operations arose frequently in the domain of diagnosis/prognosis generation problem domain. Since then it has become apparent that the facilities provided are applicable to problems both within and outside of AI. The motivation about LIMAP, its implementation, and its capabilities are presented in appendix A.

Using LIMAP the functional dependencies are represented in a symbolic matrix. Figure 4.2 shows the functional dependency graph for the engine depicted in figure 4.1. Figure 4.3 depicts the adja-

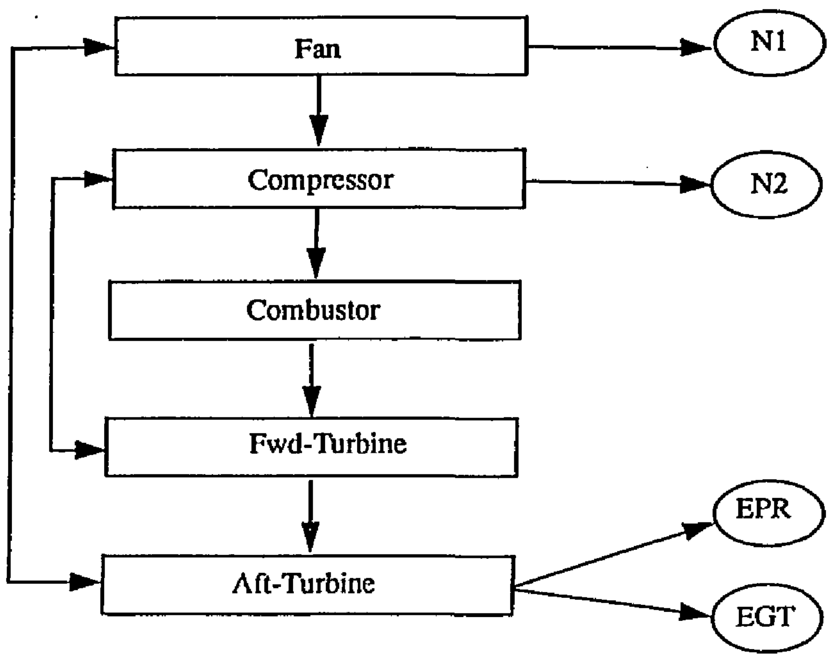

Figure 4.2: Functional dependency graph of an engine

cency matrix representing the jet engine functional dependency predicate Engine(x,y) of Figure 4.2 over the domain Comps={fan, compressor, combustor, fwd-turbine, aft-turbine, N1-sensor, N2-sensor, EGT-sensor, EPR-sensor}. A value of 1 in location i,j represents the fact that component i is connected to component j.

COMPONENT		(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Fan	(0)	.	1	.	.	1	1	.	.	.
Compressor	(1)	.	.	1	1	.	.	1	.	.
Combustor	(2)	.	.	.	1	.	.	.	.	.
Fwd-turbine	(3)	.	1	.	.	1	.	.	.	.
Aft-turbine	(4)	1	.	.	.	.	.	.	1	1
N1 Sensor	(5)	.	.	.	.	.	.	.	.	.
N2 Sensor	(6)	.	.	.	.	.	.	.	.	.
EGT Sensor	(7)	.	.	.	.	.	.	.	.	.
EPR Sensor	(8)	.	.	.	.	.	.	.	.	

Figure 4.3: Adjacency matrix for jet engine depicted in figure 4.1

4.3.4 Physical Dependency Model

The physical dependency model is a digraph of an aircraft system, similar to the functional dependencies digraph, in which the links in the graph represent potential paths of fault propagation due to physical proximity. This sort of propagation occurs when uncontrolled discharges of energy attendant on component malfunctions propagate to neighboring systems. The severing of nearby hydraulic lines by blade fragments from a disintegrating turbine provides a typical example. As with the Functional Dependency Model, this model is also implemented using LIMAP.

4.3.5 The Abstraction Hierarchy

The case-based reasoning component of Epaion consists of a self-organizing memory structured as a frame-based abstraction hierarchy, as defined by [Schank 1982]. This memory forms an upper bounded semi-lattice that contains domain specific information at different levels of abstraction. The information contained in the lattice includes:

a. The names of all the components in an aircraft engine.

b. The components that are sensors. The exhaust gas temperature, the rotational speed of the fan, and the fuel flow indicator are some of the mechanical sensors in an aircraft's engine. Vision, sight, and smell are the "human sensors" used in the diagnostic process.

c. The possible values for each sensor. For a mechanical sensor the allowable values are: lower than expected; normal; higher than expected. If a sensor initially indicates values that are normal, then at the following time interval indicates values that are lower than expected, and at the third time interval still indicates values which are lower than expected, then the status of the sensor during these three time intervals is normal, lower, lower which is a kind of (i.e., a subcategory of) overall lower than expected status which in turn is a kind of abnormal status.

d. The various faults that may be observed in an engine subsystem. For example it is represented that seagull ingestion is a kind of bird ingestion fault which is a kind of foreign object ingestion fault and so on.

e. Information on how faults manifest themselves. For example, the fan vibration and abnormality in the rotational speed of the fan are manifestations of a problem in the fan.

f. The accident/incidents that the system already knows. For example the system knows that the incident of a China Airlines Boeing 747 that suffered a mishap over the Pacific Ocean on February 19, 1985 [NTSB-AAR-86-03] is an instance of an accident/incident since it is a kind of rotor related scenario which is a kind of engine related scenario which is a kind of accident/incident scenario.

4.4 Reasoning Cycle

Epaion's reasoning cycle consists of the following three phases: input a new problem; retrieve the most similar case; adapt the retrieved case to fit the current scenario.

4.4.6 Case Matching and Retrieval Process

When the system experiences a new set of symptoms, i.e., when faced with an input (new) case, it searches its case library for the most similar case. This is done by placing the input case in self-organizing MOP memory under the most appropriate parents, determined as described in Chapter 2. The siblings may therefore be assumed to be closely related. The nearest sibling is retrieved as the case that is most on-point with respect to the input case.

A weighted count of corresponding symptoms between the input case and its siblings in the case library is used as a metric of similarity between the input case and each sibling. Based on the observation that in most cases similar faults manifest themselves in similar ways only during the first moments of the fault occurrence [1], the system takes advantage of the available temporal information in each case, and gives higher degree of similarity to symptoms that manifest itself in similar ways during the first moments of the fault occurrence. For example, if in the input case the rotational speed of the fan was initially abnormally high, then normal, and at the end was abnormally low, then a library case where the rotational speed of the fan was initially abnormally high and continued to be high through out the entire scenario will get a higher degree of similarity comparing to another library case where rotational speed of the fan was initially abnormally low and continued to be low through out the entire scenario.

In addition to the set of symptoms, Epaion takes into consideration the context variables of each case. For example, if the input case represents a scenario where an aircraft was flying at a high altitude then this is taken into consideration to give smaller degrees of similarity to library cases where the cause of the fault was bird ingestion. This is in accordance with the fact that birds do not fly at high altitude.

4.4.7 The Case Adaptation Process

When the system finds and retrieves a similar case, Epaion assumes that the current fault is the same as the fault in the retrieved case and adapts the causal explanation of the retrieved case to fit the current case. Then both the fault and the causal explanation are stored in the case library for future usage. The system is provided with a set of adaptation rules which, in addition to adapting the retrieved causal explanation to fit the current case, find possible gaps in the causal explanation and fill in the missing causalities by using the model. This causal explanation connects the symptoms to a justifying cause, and thus the system's causal reasoning ability produces a causal analysis of the new case, rather than simply a reference to a previous solution. The new causal analysis is not only stored in the case library as part of the input case, but is used to augment and modify the causality knowledge of the world model. The causal analysis consists of a sequence of pairs of the type ``event A causes event B", ``event B causes event C" and so on. Each of these pairs is stored in the database of the causal submodel. In the case that the model already knows about the causal relation between two events from a previously encountered case, the system updates the frequency count between the two events. The world model is therefore created based on the previous behavior of the physical system, and is constantly updated based on the current behavior, either by augmenting its previous causal knowledge or ``becoming more sure" about causal relations.

Epaion's adaptation algorithm is summarized in the following two steps:

The first step involves the transfer of the fault from the library case in the input case and consists of two possibilities.

Case 1: If the match between the input case and the library case exceeds a threshold value then the fault is transferred intact, thus if in the library case the fault was a malfunctioning fuel controller then it is assumed to be the same in the input case.

Case 2: If the match is below the threshold value then an abstraction of the library case fault is transferred to the input case. For example, if in the library case the fault was bird ingestion, then it is assumed that in the input case the fault is foreign object ingestion.

The second step involves the adaptation of the causal explanation of the library case so it can explain every, or as many as possible, of the symptoms of the input scenario by connecting them to a justifying cause. This consists of the following possibilities:

Case 1: If the library case and the input case have identical symptoms then the causal explanation of the library case is transferred intact to the input case.

Case 2: If the input case contains symptoms that do not appear in the library case then the causal explanation of the library case is transferred in the input case and the following additional processing takes place. Let ϕ_2 be an unexplained input case symptom.

Subcase 1: If the causal submodel contains the relation ϕ_1 *causes* ϕ_2, and ϕ_1 is a symptom or manifestation in the input case, then the link ϕ_1 *causes* ϕ_2 is added in the causal explanation of the input case.

Subcase 2: The causal portion of the model does not contain the relation ϕ_1 *causes* ϕ_2, but the functional dependency submodel knows that component C2 is functionally dependent on component C1, and ϕ_1 is a manifestation of abnormal behavior of component C1, and similarly ϕ_2 is a manifestation of C2. This knowledge is depicted by the graph

$$\begin{tikzcd}
\phi_1 & & \phi_2 \\
C_1 \arrow[u] \arrow[rr, "D"] & & C_2 \arrow[u]
\end{tikzcd}$$

Figure 4.4: A causal scenario

where ϕ denotes a phenomenon that is a symptom or manifestation (μ) of abnormal behavior of a component. Additionally, if ϕ_1 is a symptom in the input case and time(ϕ_1) $\leq$ time(ϕ_2), i.e., symptom f_1 appeared before ϕ_2 then the link ϕ_1 *causes* ϕ_2 is added in the causal explanation of the input case.

Case 3: If the library case has symptoms that do not appear in the input case then the causal explanation of the library case is transferred in the input case and the following additional processing takes place. Let ϕ_2 be such a symptom in the library case. Then the causal explanation of the library case will contain the relation ϕ_1 *causes* ϕ_2.

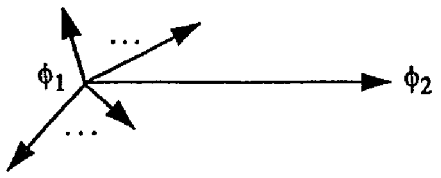

Figure 4.5: Relation ϕ_1 causes ϕ_2

Subcase 1: Suppose that this configuration occurs in the functional portion of the model.

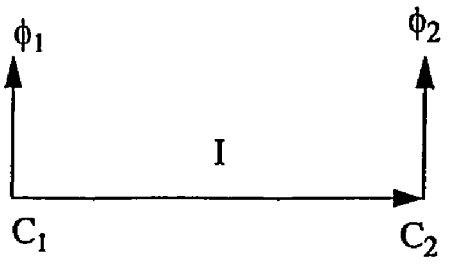

Figure 4.6: A causal scenario

Then this library case is rejected as explanation of the input case since if C1 were in fact abnormal in the input case, then the immediate link between C1 and C2 indicates that this malfunction must propagate immediately to C2, and therefore a manifestation of C2's abnormality would be present. But the input case shows no such manifestation, so C1 is normal.

Subcase 2: Suppose that this configuration occurs in the functional portion of the model.

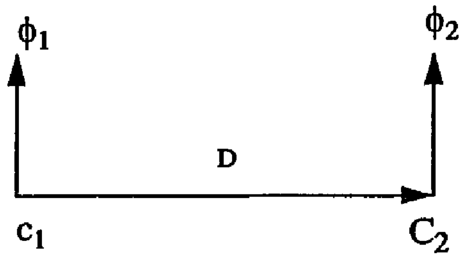

Figure 4.7: A causal scenario

where ϕ_2 is the unmatched library case symptom and D is a non immediate link between component C1 and C2. Then the library symptom ϕ_2 is ignored, since it is possible that ϕ_2 will occur later in the library case. Therefore every relation of the form X *causes* ϕ_2 is discarded from the transferred causal explanation.

At this point Epaion has used knowledge of how faults manifest themselves, knowledge of causal links between fault manifestations, and knowledge about links between components to explain as many of the symptoms that are present in the input case. Any additional symptoms will remain unexplained.

Chapter 5

Evaluation

5.1 Introduction

The diagnostic methodology presented in this thesis was evaluated by means of an analytical analysis of the methodology and by an empirical analysis of the prototype developed to support the merit of the methodology. In the analytical evaluation we describe the characteristics from which the methodology draws its power, and discuss the consequences of incompleteness or elimination of the four knowledge sources that are involved in this methodology. The empirical analysis evaluates the prototype that was build in order to support the merit of the methodology. This evaluation was done by running Epaion on actual accident cases and comparing the results with the conclusions of the official investigations on these accidents.

Before the evaluations we present two examples in order to demonstrate how Epaion works. The first example involves two realistic scenarios. EPAION was given the symptoms observed on January 8, 1989 by the flight crew of British Midland Airways and retrieved as the most on-point case scenario the Overseas National Airways flight 32 crash that occurred on November 12, 1975. For the second example Epaion was given a complex hypothetical scenario. This example demonstrates Epaion using all of its knowledge sources, the library case, the functional dependency submodel, the causality submodel, and its abstraction hierarchy in order to connect all of the observed symptoms in the hypothetical scenario to a justifying cause.

5.2 Examples

The ideas presented in this thesis are demonstrated by the following examples.

5.2.1 First Scenario

EPAION is given the symptoms observed on January 8, 1989 by the flight crew of British Midland Airways. The senario is summaried as follows:

> The plane, a Boeing 737-400, was climbing out east of East Midlands Airport and the crew operated at a high workload. On this flight, the crew experienced severe vibration and smoke in the cockpit. The vibration monitor on the left engine.was at the high value, while the rotational speed of the fan (N1) was fluctuating. Subsequently, the exhaust gas temperature (EGT), N1, and rotational speed of the compressor (N2) reached high levels. The fuel flow was low. The vibration continued to be severe and the fuel flow low, but N1 and N2 diminished to low levels.

Epaion's first task is to use the features of the current situation for finding the most similar scenario from its case library. In this example the retrieved scenario is the Overseas National Airways flight 32 crash that occurred on November 12, 1975. The scenario is summarized as follows:

> The plane, a DC-10-30, was taking off from John F. Kennedy International Airport, in New York. The crew observed that the rotational speed of the fan (N1) was fluctuating, along with the rotational speed of the compressor (N2) and exhaust pressure ratio (EPR). Later on N1 and N2 were increasing to high while the exhaust gas temperature (EGT) and the EPR were

increasing to high also. The Fuel-Flow began fluctuating. Finally N1, N2, and Fuel-Flow started decreasing and reached low levels. The EGT continued to increase.

The Overseas National Airways crash was retrieved as the most-on-point case for the British Midland Airways scenario because of high degree of similarity in the behavior of the fuel flow, the rotational speed of the fan, the rotational speed of compressor and the exhaust gas temperature. The behavior of the exhaust pressure ratio (EPR) was not taken into consideration since the engines on the Midland aircraft were General Electric CFM56s, which have no EPR sensor. Other cases in the library that demonstrated similarities in features such as the type of the airplane, the type of the engines, the airline, the phase of the flight, the altitude, etc., were not retrieved since similarity in these features is considered less significant.

Following the retrieval, EPAION assumes that the cause of the symptoms in the current situation is the same as the one in the retrieved case. In the Overseas National crash the cause was fan blade damage and the system tries to explain as many as possible of the Midland symptoms based on that cause. This is done by adapting the causal explanation of the Overseas National case to fit the current situation.

In the Overseas National case a large number of sea gulls were ingested into the engine causing the engine to disintegrate. The disintegration resulted in abnormalities in the rotational speed of the fan (N1) and the rotational speed of the compressor (N2). Abnormality in N2 caused the abnormal behavior of fuel flow and the high levels of EGT. In turn the fuel flow abnormality caused the EPR abnormality. This chain of events explains the behavior of N1, N2, EGT and fuel flow in the Midland scenario but does not explain the fan vibration experienced by the Midland flight crew.

In order to explain the fan vibration EPAION utilizes its models. The causal model informs the system that based on previous cases the system has learned that the leading (most often observed) cause of fan vibration is fan blade damage. Based on that knowledge the system explains Midlands fan vibration as a result of the fan blade damage. Since all of the Midland symptoms have been explained, EPAION creates the causal explanation for Midland by connecting each symptom to its cause. This causal explanation is associated with the Midland accident scenario and is stored in the case library.

5.2.2 Second Scenario

We assume that EPAION is given the following data:

> The plane is climbing out, with the crew operating at moderate workload. The engine commanded status is at climb power. The weather is icing. The crew observes a small thrust shortfall and vibration in the compressor and fan rotors. The compressor rotor speed (N2) shows a 5% shortfall. The exhaust gas temperature (EGT) and fuel flow are slightly lower than expected.

0i Epaion's first task is to use the features of the current situation for finding the most similar scenario from its case library. In this example, the selection process results in retrieval of the following case:

> The plane was climbing out, with the crew operating at a moderate workload. The engine commanded status was at climb power. The meteorological conditions were icing. Fan blade damage, caused by ice ingestion, produced an abrupt change of vibration in the fan rotor and abnormality in the rotational speed of the fan (N1).

Following the retrieval, EPAION assumes that the cause of the symptoms in the current situation is the same as the one in the retrieved case. In this example the cause is ice ingestion and the system tries to explain all or most of the current symptoms based on that cause. This is done by adapting the causal explanation of the retrieved case to fit the current situation.

In the retrieved case ice ingestion caused imbalance of the fan rotor, which in turn caused the fan to vibrate and rotate at abnormal speed. This chain of events explains the fan vibration in the current situation, but does not explain the abnormalities in the speed of the compressor rotor, the EGT, the fuel flow and the compressor vibration.

In order to give explanations for these symptoms EPAION utilizes its models. The causal model informs the system that based on previous cases the system has learned that the leading cause of abnormal speed of the compressor rotor is abnormal vibration of the fan and the leading cause for abnormal EGT and fuel flow is abnormality in the speed of the compressor rotor. In addition, the causal submodel informs the system that the leading cause of abnormal thrust output is vibration of the compressor. Based on that knowledge the system explains the current abnormality in the compressor rotor speed as a result of the abnormal fan vibration and the low levels of EGT and fuel flow as a result of the compressor rotor speed shortfall. The thrust shortfall is explained as a product of the compressor vibration.

At this point all of the current symptoms are explained except for the compressor vibration. The system from its knowledge contained in the abstraction hierarchy knows that vibration of the compressor is a manifestation of abnormal behavior of the engine's compressor. The functional model knows that the compressor is functionally dependent on the fan and therefore tries to find if any of the manifestations of abnormal fan operation are being experienced by the crew. Fan vibration is one of the current symptoms and is a manifestation of abnormal fan operation, therefore the system explains the compressor vibration as a product of the fan vibration. The set of symptoms of the retrieved scenario contains an abnormality in the rotational speed of the fan. This is not experienced during the current situation, therefore no further explanations are needed.

As soon as all symptoms are explained, EPAION creates the causal explanation of the current case by connecting each symptom to its cause. This causal explanation is associated with the current situation, and is also stored in the case library for future reference. Figure 1 displays the chain of causal events in the retrieved and the current case.

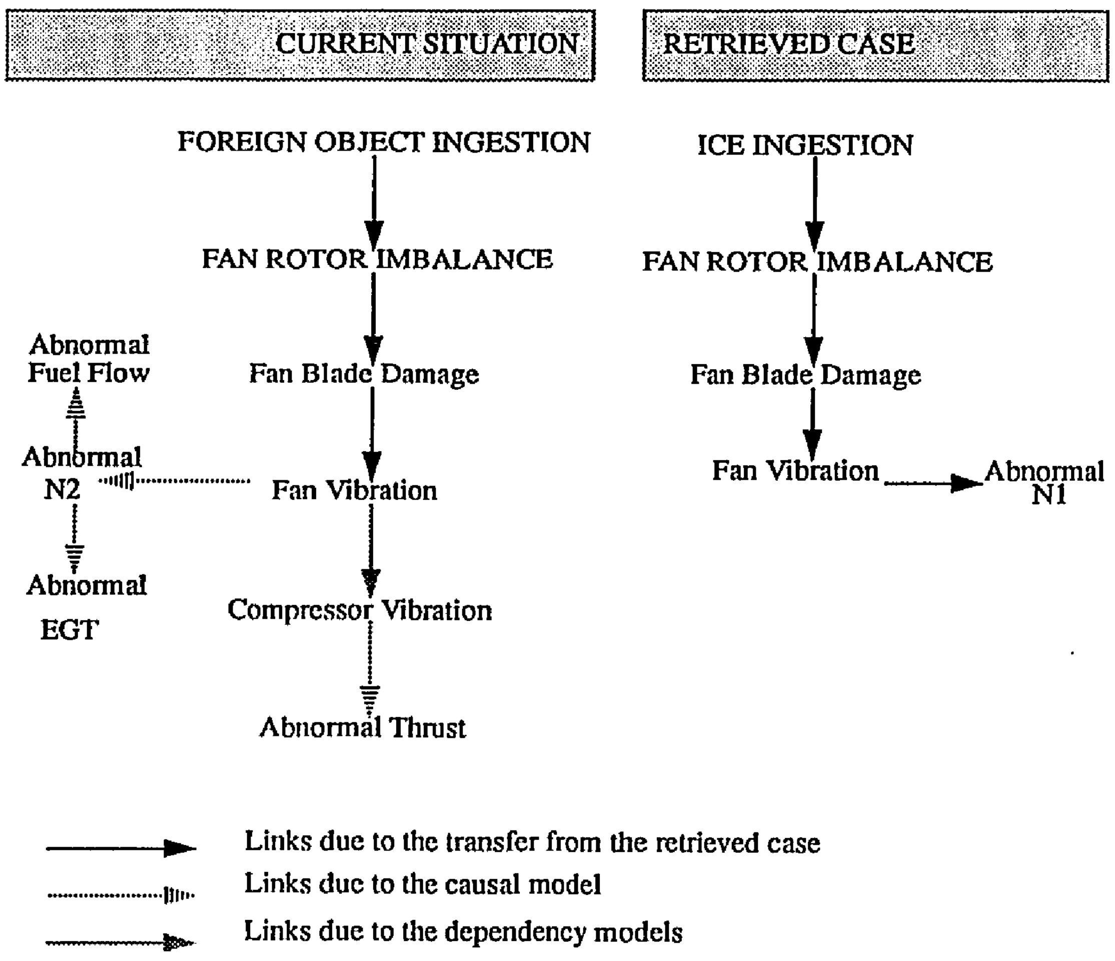

Figure 5.1: Causal explanations of retrieved and current case

5.3 Analytical Evaluation of the Methodology

The analytical analysis of the diagnostic methodology presented in this thesis involves a description of the characteristics from which the methodology draws its power, and a discussion of the consequences of incompleteness or elimination of the necessary knowledge that this methodology requires. This analysis is domain independent and applies to any reasoner that will attempt to reason about physical systems within the framework of the diagnostic methodology presented in this thesis.

Epaion's diagnostic methodology draws its power from the following four knowledge sources: the library case, the functional dependency submodel, the causality submodel, and the abstraction hierarchy. In this section we describe the important characteristics of each knowledge source, together with the consequences of not possessing these characteristics.

5.3.1 The Case Library

The methodology presented in this thesis requires that the reasoner maintain a library of previously solved problems. Each problem is a description of a physical system malfunction and the manifestation of the malfunction. For example, Epaion's case library consists of actual aircraft accident/incident scenarios. Information provided in the individual accident/incident reports from the National Transportation Board (NTSB), the British Air Accidents Investigation Branch (AAIB), and data collected from test accidents staged at Boeing Inc. [Shontz et. al. 1992] was used to derive the appropriate information constituting each case.

Case Description

Each malfunction is described as a set features. Each feature has an associated set of possible values. The features are clustered into the following five categories: Identification Features, Context Variables, Symptoms, Fault, Causal Explanation. The choice of the features is done by taking in to consideration:

a. If a particular feature is unique in the sense that the value of that feature identifies one and only one case, then this feature should be included as an identification feature.

b. If a particular feature does not have a unique value but the value of the feature may help the human operators of the physical system to be reminded of an actual case that they happen to have directly or indirectly witnessed, then this feature may be included as an identification feature. For example, Epaion includes as identification features the features *airline* and *date*. These features do not have unique values for each case but collectively may remind pilots about a particular accident or incident.

c. If the physical system includes a mechanical sensor that monitors the behavior of a particular component, then each case must include a feature that describes the behavior of the sensor.

d. If there are is an event that may be witnessed by the human operators of the physical system, then each case must include a feature that describes the presence or absence of this event.

Whenever one of the above conditions holds, the corresponding feature must be be included in a particular case. Exceptions are the fault and causal explanation features that every case must include. Each case must include a feature that reveals the fault in the particular case. In addition a feature should be included that describes the chain of events that connects each observable symptom to a justifying cause. When the *fault* and *causal explanation* features are missing the reasoner's task is to find a value for those two features.

5.3.2 The Functional Dependencies Submodel

The functional dependencies submodel possesses two kinds of constituents: components, and interconnections between components.

Interconnections

Functional dependency links represent all the potential paths of normal interaction between components in the physical system. When a fault occurs, the effect of the fault is expected to propagate along one of the paths in the functional dependencies model. Whether or not a normal interaction occurs along a particular path may depend on specific parameters that are unavailable in the model. By representing all potential of normal interaction we can represent even those fault cases where the interaction is not anticipated under the current scenario but happens unexpectally [Abbott 1990].

Definition and Choice of Components

A component is a physical part or set of parts of the particular physical system that the reasoner is called to reason about. A component may consist of other components which in turn may have subcomponents, etc. A physical system may have a varying number of components, depending on the level of detail at which we view the system. Choosing the appropriate level of detail means choosing which components we need to include in the model of the physical system so we can have appropriate diagnoses of abnormal behavior. The choice of components is done by taking in to consideration [Abbott 90]:

a. Whether a particular component must be identified as faulty when it breaks. If it is important to identify when a component fails, say because the manifestation of the failure may be apparent to the operators of the physical system, then this component should be included in the model.

b. Whether a component can be disambiguated with the available sensors.

c. Whether a particular component is needed in the propagation path to determine the propagation of abnormal behavior. If a particular component is a branching point in the propagation path that enables identification of the propagation to other components then it must be included in the model.

If none of the above factors holds, then a component either should not be included in the functional dependencies submodel, or aggregating it to the next higher level of detail should be considered. For example, in Epaion's functional dependencies submodel individual fan blades are not included in the model of an aircraft's engine because it makes no difference if blade 8 or blade 9 fails. An additional reason the submodel does not include fan blades is that there is no sensor information to identify individual fan blades. On the other had, by aggregating the fan blades to the next higher level of detail the engine's fan is included in the submodel.

When a component fails, the reasoning system will be aware of the manifestation of this failure. If one of the above factors holds but the component is not modeled in the functional dependencies submodel then the reasoning system will not have the ability to link this manifestation to a justifying cause since it will have no knowledge of functional dependency between the component that is not modeled and other components in the submodel.

5.3.3 The Physical Dependencies Submodel

Similarly to the functional dependencies submodel, the physical dependencies submodel possesses two characteristics: components, and the interconnections between components.

Interconnections

Physical dependency links represent potential paths of fault propagation that are due to physical proximity. This knowledge is contained in a graph representation similar to the representation of functional dependencies. The edges of the graph represent the physical proximity links and the nodes represent components.

Definition and Choice of Components

Components in this submodel are defined as in the functional dependency submodel and the criteria for choosing which components we need to include in the physical submodel are the same with the criteria presented above.

5.3.4 The Causal Dependencies Submodel

The characteristics of the causal dependencies submodel are: the events, and the relationship between the events.

Causal Relationship

A link between two events e_1 and e_2, indicates that one did cause the other (say e_1 caused e_2). The causality relationship between e_1 and e_2 implies a temporal constituent: If e_1 occurred at time t_1 and e_2 at time t_2 then $t_1 < t_2$.

Definition and Choice of Events

An event is a qualitative state transition to an abnormal state in the behavior of a physical system. Events may be witnessed either by observing the behavior of mechanical sensors or by stimulating "human sensors" such as sight, smell, and hearing. The choice of the events is done by taking in to consideration:

a. If the physical system has a mechanical sensor that monitors the function of a component or a process, then an event signaling the abnormal behavior of the function or the process must be included in the set of events.

b. If there is occurrence that may be witnessed by the human operators of the physical system, then an event signaling the occurrence must be included in the set of events. Such occurrences include the smell of smoke, visibility of fire, hearing an explosion, etc.

If one of the above factors holds but an event is not included in the list of events, then the reasoner loses its power of justifying the observable symptoms in a new situation precisely and in detail. Additionally, when the new situation is stored in the case library it becomes a "weak" most-on-point case for a potentially similar future input case.

5.3.5 The Abstraction Hierarchy

The information contained in the abstraction hierarchy must include:

a. The names of all the components in the physical system

b. The components that are sensors.

c. The possible values for each sensor.

d. The types of faults that may effect the physical system.

e. Information on how faults manifest.

f. The cases that the reasoner experienced.

All this information should be represented at different levels of abstraction. The levels of abstraction chosen must be determined by examining the domain itself, and what information the human operators of the physical system might use to make decisions.

The choice of components and sensors is made based on the criteria followed for the choice of components in the dependency models. Experimental observation has proved that the most effective allowable values for each sensor are qualitative descriptions of the sensor readings. The range of these values is the following enumerated set: normal, lower than expected, higher than expected, fluctuating. The types of faults that may effect the physical system along with information on how faults manifest is domain dependent and may be elicited from domain experts.

An incomplete abstraction hierarchy may affect the reasoner's capability to explain the presence of the symptoms experienced in the current situation. The empirical evaluation of Epaion, as it is presented in the following section, serves as an example of the effects of incomplete knowledge in the reasoner's abstraction hierarchy.

5.4 Empirical Evaluation of the Prototype

This section describes an empirical evaluation of the diagnostic concepts implemented in Epaion. The evaluation uses actual aircraft accidents and incident cases, which were simulated to assess the effectiveness Epaion in diagnosing failures.

5.4.1 Approach

Epaion was developed using a software engineering strategy known as incremental code revision or rapid prototyping. Rapid prototyping requires the incremental development of the software design to be guided by preliminary evaluations of the software. Our evaluation approach consisted of comparing Epaion's output to the "correct answer" in order to determine how well the program has performed.

Information provided in the individual accident/incident reports from the National Transportation Board (NTSB), the British Air Accidents Investigation Branch (AAIB), and data collected from test accidents staged at Boeing Inc. [Shontz et. al. 1992] was used to derive the appropriate information constituting each case, a process called accident reconstruction. We reconstructed a total of eighteen cases, of which twelve were used as library cases, and six as input cases.

Accident reconstruction is not a straightforward process and has its limitations. In the reconstruction process the symptoms from all accidents had to be identified from the sources that described the accidents. Unfortunately numerical sensor data from the engine parameters was not available, so the symptoms were used as reported in [Shontz et. al., 1992], or derived based on the descriptions in the NTSB or AAIB analysis of each accident. NTSB and AAIB reports did not always explicitly describe the symptoms in each case; even in those cases where symptoms were mentioned explicitly they were usually only those described by the flight crew. The sequence of symptoms could therefore not always be determined completely.

In addition a chain of causalities had to be constructed for each of the accidents used as library cases. This chain explains each observed symptom by connecting the symptom to a justifying cause. Determining the causal explanation of the symptoms for each case was a difficult task because of a paucity of definitive experts who could provide this information. While pilots, maintenance personnel, and aircraft system designers are all knowledgeable about some aspects of aircraft diagnosis, each has deficiencies in one area or another. The causal explanations used in each library case were constructed after interviewing personnel with expertise in the above fields, and consulting NTSB and AAIB reports.

The evaluation process required that each input case be presented to Epaion separately, and that the system produce a diagnosis along with a causal explanation. The diagnosis produced by Epaion was then compared with the correct diagnosis for the particular scenario. In addition, the reasoner was evaluated based on the number of symptoms for which the reasoner was able to find a justification. A "correct diagnosis" is the diagnosis determined by NTSB, AAIB, or by [Shontz et. al. 1992]. Epaion is said to have produced a complete explanation if the system was able to explain each observed symptom by connecting the symptom to a justifying cause.

5.4.2 Results

In this section the resulting diagnosis for each input case is presented and discussed.

Case 1:

We presented to Epaion the incident of a British Midland Boeing 737-400 (G-OBMG) that took place on June 11, 1989 [AAIB-AAR-4/90]. In this incident the aircraft was climbing when the crew reported an onset of "thumping" and severe vibration. Reference to engine instruments revealed high indicated vibration with low and fluctuating rotational speed of the fan on the N2 engine. The crew also reported that there was considerable smoke in the aft cabin, and that flames and sparks had been seen to come from the right engine. The aircraft landed without further incident. Examination of the engine after landing showed that the fan had been massively damaged.

Epaion correctly classified the incident as a case involving a rotor damage. Epaion already had in its library two other scenarios in this category: an incident involving a Dan Air Boeing 737-400 that took place on June 9, 1989 [AAIB-AAR-4/90], and the accident of a British Midland Boeing 737-400 that took place on January 8, 1989 [AAIB-AAR-4/90]. Both library cases achieved a high degree of similarity with the current scenario, but the latter case achieved the highest degree of similarity, and therefore was retrieved as the most-on-point case for G-OBMG. The fault of the retrieved case was assumed to be the fault in the current case, thus Epaion correctly diagnosed that the problem in the current scenario was fan blade damage. The causal explanation of the retrieved case was transferred to the current case, and since in both cases the fault manifested itself in a very similar way (similar symptoms though time), the transferred causal explanation was able to successfully explain every symptom experienced in the G-OBMG.

Case 2:

Epaion was presented with the incident of an American Airlines Boeing 727 (Flight 566) that experienced a engine failure in its number one engine just after rotation on take-off from Greater Cincinnati Airport, Cincinnati, Ohio [NTSB-78-F-A067]. The captain performed emergency shutdown procedures on the engine and returned to the airport. The NTSB determined that the engine failure was caused by several turbine blade separations.

Epaion correctly classified the incident as a scenario involving a rotor failure. Among four other cases under this category it retrieved as the most on-point case the above-mentioned incident of June 9, 1989, involving a Dan Air Boeing 747-400. The fault of the retrieved case was assumed to be the fault in the current case, thus Epaion incorrectly diagnosed that the problem with the current scenario was fan blade damage instead of turbine blade damage. The transfer of the causal links from the retrieved causal explanation was sufficient to explain all the symptoms in the American Airlines case except the abnormal exhaust pressure ratio. After consulting the causal model the system produced explanations for all the symptoms.

Case 3:

The incident of a China Airlines Boeing 747 (Flight 006) that suffered a mishap over the Pacific Ocean on February 19, 1985 [NTSB-AAR-86-03] was presented to Epaion. The aircraft was cruising on autopilot when the crew misdiagnosed a flame-out in the number one engine. In reality another engine had a bad fuel controller and suffered a condition known as bleed-air hogging. The bad fuel controller caused a flame-out and due to a series of misdiagnoses and inappropriate corrective actions by the crew the aircraft was put into a vertical dive. Finally the captain regained control of the aircraft and made a safe landing in San Francisco.

Epaion correctly classified the incident as a scenario involving a fuel subsystem failure. The library contained two other scenarios in this category. A Boeing test case involving a bad fuel metering unit was retrieved as the most on- point case. Because the degree of similarity between the China Airlines scenario and the retrieved case was not very high, Epaion correctly assumed that the current fault was an abstraction of the fault in the retrieved case, and determined that the fault was in the fuel subsystem. The transfer of the causal links from the retrieved causal explanation was sufficient to explain all the symptoms in the current scenario except the abnormal behavior of the Exhaust Pressure Ratio (EPR). By consulting its world model Epaion found that the abnormal EPR was due to the abnormality in the fuel flow, thus successfully explaining every observed symptom.

Case 4:

In June 1982 the Galunggung Volcano on the island of Java erupted. A Boeing 747 encountered the volcanic debris and experienced flame-outs on three engines while the aircraft was at 33,000 feet. One engine was successfully restarted and an uneventful two-engine landing was accomplished [Lloyd 1990].

By relying on observations from "human sensors", when Epaion was presented with this scenario it successfully classified the incident as a volcanic ingestion scenario. The systems case library contained two scenarios under this category. An incident of volcanic ingestion experienced by a Boeing 747-400 near Anchorage, Alaska on December 14, 1989 [Lloyd 1990] was retrieved as the most on-point case. In both the input and the retrieved case the set of symptoms over time was almost identical, and therefore Epaion correctly determined the fault and produced a causal explanation that covered all of the symptoms experienced in the Galunggung incident.

Case 5:

The accident of a Southern Airways DC-9 (Flight 242) that crashed in New Hope, Georgia on April 4, 1977 [NTSB-AAR-78-3] was presented to Epaion. The aircraft had flown through heavy thunderstorms and had lost both engines. The NTSB determined that massive water ingestion into the engines accompanied by thrust lever movement induced severe stalling in, and major damage to, the engine compressor.

Epaion's case library had no previous case of massive water ingestion. The system classified the accident in the category of "miscellaneous scenarios", and retrieved as the most on-point case the accident of an Overseas National Airways DC-10-30 that took place on November 12, 1975 [NTSB-AAR-76-19]. The fault in the retrieved case was bird ingestion. Because the retrieved case did not achieve a high degree of similarity with the Southern Airways case the system correctly assumed that the current fault was an abstraction of the fault in the retrieved case. Epaion determined that the current fault was foreign object ingestion. The transfer of the causal links from the retrieved causal explanation was sufficient to explain completely all the symptoms in the Southern Airways case.

Case 6:

The symptoms observed during Boeing's test flight F5 [Shontz 1992] were presented to Epaion. This was a case of heavy damage due to ice ingestion. Epaion correctly classified the case as an icing scenario. Under this category the case library had two other scenarios. A scenario of moderate ice ingestion was retrieved as the most on-point case, and based on that scenario the system correctly assumed that the fault in the input case was ice-ingestion.

The transfer of the causal explanation from the retrieved case to the input case was sufficient to explain all the symptoms in the input case except for the abnormal behavior of the rotational speed of the fan and the presence of broad-band vibration. Both of these symptoms were absent from the retrieved case, since the retrieved case was an instance of moderate ice ingestion, whereas the input case was an instance of heavy ice-ingestion. By utilizing the causal dependencies portion of its model, Epaion was able to explain that the abnormal behavior of the rotational speed of the fan was attributed to the abnormality of the fuel flow. Lack of relevant knowledge in the systems' causal submodel and abstraction hierarchy made Epaion unable to explain the presence of the broad-band vibration.

Table 1 presents a summary of the results. The first two columns identify each scenario that was presented to Epaion as an input case. The following two columns identify the appropriate classification of the accident/incident along with the actual fault as determined by either the NTSB, the British Air Investigations Branch, or Boeing's test data. The fifth and sixth columns present the classification of each accident/incident done by Epaion along with the fault assumed by Epaion.

The last column tabulates the result of Epaion's adaptation phase. Epaion's explanatory performance was characterized as complete in the cases where the system was able to causally justify every symptom experienced in the input case.

	Case Identification	Correct Classification	Correct Fault	Epaion's Classification	Epaion's Fault	Epaion's Explanation
1	G-OBMG	Rotor Scenario	Fan Blade	Rotor Scenario	Fan Blade	Complete
2	American Airlines 566	Rotor Scenario	Turbine Blade	Rotor Scenario	Fan Blade	Complete
3	China Air 006	Fuel Scenario	Fuel Controller	Fuel Scenario	Fuel Subsystem	Complete
4	Galunggung	Volcanic Scenario	Volcanic Ingestion	Volcanic Scenario	Volcanic Ingestion	Complete
5	Southern Airways 242	Water Scenario	Water Ingestion	Miscellaneous Scenario	Foreign Object Ingerstion	Complete
6	Boeing Test Flight F5	Icing Scenario	Ice Ingestion	Icing Scenario	Ice Ingestion	Incomplete

5.4.3 Discussion

Automation of inflight dia- and prognosis as an aid to the flight crew has great potential for improving the general safety of civil transport operations. The Epaion case-based reasoning system we have developed for the purpose of performing fault diagnosis and prognosis of aircraft in operation uses a hybrid reasoning process based on a library of previous cases and several models of the aircraft as basis for the reasoning process. This arrangement provides the methodology with the flexibility and power of first-principle reasoners, coupled with the speed of associational systems.

We have evaluated the system's performance empirically on six actual accidents/incidents. The results achieved are very promising for the future success of the system. Based on the results we make the following observations.

* Classification

Five of the six cases in this evaluation were correctly classified. Case No. 5, involving water ingestion, was classified under the category of miscellaneous scenarios due to the lack of previously encountered water ingestion scenarios. This actually can not be considered misclassification since it is expected that scenario types that were not encountered by the system will classified as miscellaneous scenarios. This suggests that an expanded case library will enhance the systems classification capability and therefore offer better matches for each additional input case.

* Diagnosis

Epaion was able to correctly diagnose five of the six scenarios. The American Airlines Flight 566 scenario (case 2) was properly classified as a rotor scenario but misdiagnosed as fan problem rather than turbine problem. This is a result of the fact that problems in the fan and problems in the turbine manifest themselves similarly, and therefore both kinds of faults are classified under the category of rotor scenarios. When case 2 was used as input case the system retrieved as the most on-point case the Dan Air incident, which is a fan blade scenario. With almost negligible difference in the degree of match between the input case and the relevant library cases, the second best match was the accident of a United Airlines Flight 611 that took place on July 19, 1970 [NTSB-AAR-72-9]. This is a turbine fault scenario and would have achieved a higher degree of similarity with the input case if the time order of the symptoms in both cases had been represented more precisely. All symptoms used in reconstructing the case of the United Airlines Flight 611 were based on expert opinion, but none was explicitly stated in the NTSB report. With the exception of the behavior of the EGT, the same holds for the symptoms used to reconstruct the American Airlines Flight 566 scenario. This suggests that presenting the system with cases which are reconstructed based on an accurate set of symptoms is vital for correct matching and therefore correct diagnoses.

* Symptom explanation

In five of the cases presented as input Epaion was able to explain all of the symptoms experienced. Failure to explain the presence of broad-band vibration in the last case (case 6) is attributed to limited information in the abstraction hierarchy. If the fact that broad-band vibration is a manifestation of fan abnormality had been included in the abstraction hierarchy, the system's functional dependencies model would have explained the broad-band vibration symptom as a result of fan blade damage. The same result would have been achieved if the system had previously experienced other cases with broad-band vibration, thus enabling the causal submodel to explain the vibration. It is evident that the more knowledge that the system contains in its abstraction hierarchy, the better its explanation capability will be. Current efforts are accordingly focused on expanding this knowledge to have a substantial size.

Chapter 6

Discussion

6.1 Contributions

[Michie 1971] presents the following criteria that constitute necessary conditions for a program to be characterized as intelligent.

- The program should utilize a model of its task environment.
- The program should use its model to form plans of action to be executed in the task environment.
- These plans should include directed sampling of the task environment so as to guide execution along conditional branches of the plan.
- The program should re-formulate a plan when execution leads to states of the environment which were not predicted in the model.
- The program should utilize the record of failures and successes of past plans to revise and extent the model inductively.

The dominant characteristic of these criteria is the presence and use of a model in the program's task environment. The CBR paradigm demonstrates promising results in areas such as planning, design and memory organization, but its success is limited due to the lack of deep domain knowledge. In the few cases where CBR is used in conjunction with deep domain knowledge the techniques employed are specific to the particular application and domain.

This thesis investigates the use of models in conjunction with a CBR methodology for physical system faults, and provide useful insights into the challenges and benefits of such a hybrid reasoning methodology. To demonstrate the methodology a prototypical system has being designed and developed in the domain of aircraft faults. Actual aircraft accident cases were used and analytical and empirical results have been presented. In summary the following has been achieved :

- A mathematical formalization of the case-based reasoning paradigm has been developed. This formalization provides a precise definition and description of the CBR paradigm, in contrast to present specifications, which consist of either lengthy and imprecise verbal descriptions, or of impenetrable LISP code. In addition, the existence of a formal model opens the possibility of a theoretical treatment of the subject. To gain precision and expressiveness we developed a formal specification of the memory organization and various phases of the CBR reasoning paradigm.

- For similar reasons the model-based reasoning paradigm was formalized. The initial formalization focuses on causal models.

- A formal mathematical definition of the functions required for interfacing the CBR component with the model-based component of this reasoning methodology was provited. Such interface functions are required during the phases of matching, and adaptation.

- The methodology was tested by designing and implementing a prototypical system for dealing with physical system faults. The system entails the use of case-based methodology in conjunction with device models that describe the physical system's structural, functional, and causal behavior.

6.2 Limitations

The research presented in this thesis has a number limitations. Our methodology suffers from the fact that the set of abnormalities that the physical system may experience along with the various ways that each abnormality manifests itself must be enumerated. If the input case demonstrates an abnormality that is not predefined in the abstraction hierarchy, or if the abnormality takes a value that is not predefined, then the system is unable to reason. For example, in order for the system to proceed with its the normal reasoning cycle when the input includes the information that the rotational speed of the fan is high in the abstraction hierarchy it must be defined *rotational speed of the fan* is one of the things that can be abnormal and that *high* is one of the ways that an abnormality manifests itself. Currently the only way to compensate for this deficiency is to carefully and clearly define all the relevant characteristics of a particular domain before this methodology is applied to this domain.

An objection to our methodology might be raised in that the behavior of the physical system that the reasoner is called to reason upon is required to be represented in terms of time episodes divided into four subphases. This restriction stems from the fact that the monitoring of most physical systems is incomplete and available library cases do not contain continuous information of the behavior of the physical system. In the aircraft domain, dividing the abnormal behavior of the engine into four episodes has produced satisfactory results. Further research must investigate the issues associated with reasoning with cases of uneven duration divided into various episodes, each of which has uneven duration as well.

Another handicap of our methodology is its current inability to recognize and deal with multiple faults. Multiple independent faults in a physical system, although uncommon, are always a possibility. In the section on future work we describe a possible extension of our methodology that would recognize and deal with situations where multiple faults occur.

As with other systems which reason based on previous cases, Epaion reasons even in the event that the case library contains only one case. This becomes a problem if the only case in the library case is unrelated to the input case. A simple solution to this is to require that no case may be retrieved from the case library unless it exhibits a a level of similarity which is higher than a given threshold value. An even better approach to the problem is to never allow the system to reason unless it has been trained with several cases.

6.3 Future Work

The work presented in this thesis may be improved and extended in various ways.

6.3.1 Representing MOPs in LIMAP

Currently each case in Epaion is represented in a memory organization packet (MOP) as implemented in [Riesbeck & Schank 1989]. This implementation uses a set of tables that maintain links between MOPs such as specialization links, abstraction links, slots etc.

MOPs may be represented using the abstract data structures of LIMAP [Feyock and Karamouzis 1992]. This can be done by defining a sparse $N \times N$ symbolic matrix, where N is the number of MOPs. An entry in location i,j denotes that MOP i is the specialization of MOP j and vice verse (j is the abstraction of i). Finding the set of all abstraction MOPs for i is as easy as scanning through row i, and scanning through column j gives all the specialization MOPs of j. All other basic operations on MOPs may be based on this matrix. This representation would result in efficient operations on MOPs and efficient memory utilization. Additionally it would provide to the developers of the system with a better way to visualize the links between the MOPs and comprehend the reasoner's memory structure.

6.3.2 Prognostication

Automating the process of predicting the future behavior of physical systems is a difficult task because physical faults manifest themselves in various ways and it is difficult to enumerate all possible consequences. Current efforts to incorporate prognostication features in diagnostic systems that reason from physical system models succeed in predicting the expected course events, but are limited by the level of detail of their models [Feyock & Karamouzis 1991]. For example, a model-based reasoning system that has a model of an airplane's functional and physical connections among components may, after establishing that the fan in the left engine is the faulty component, predict that the fault will affect the operation of the compressor since there is a functional link between the two components. Such a system is incapable, however, of deducing that flying fragments from the faulty fan may penetrate the fuselage and damage the right engine. Humans, on the other hand, are good at making such predictions, since their reasoning is based not only on pre-existing models of the world, but also on previous directly or vicariously experienced events which remind them of the current situation.

The nature of reasoning that takes place in CBR systems resembles the human ability of being reminded; CBR systems therefore offer a prognostication capability similar to the one that humans demonstrate. This capability, however, is limited to the knowledge contained in the retrieved case.

The methodology presented in this thesis offers a prognostication ability that is beyond the capabilities of conventional CBR systems. This ability stems from the existence and utilization of the models. For example, having achieved a match of the current situation with a previous case where the faulty component was a malfunctioning fuel controller, the system will hypothesize that the same fault is occurring. By referencing the world model it is able to predict that an engine flameout may occur, although that did not happen in the retrieved case, because the model may have recorded at least one previous instance where this happened. The operator is provided with a list of possible consequences of the fault along with a frequency count of each one.

6.3.3 Multiple Independent Faults

This thesis offers a reasoning methodology for dealing with abnormal behavior of physical systems in operation, but always assumes that the observed symptoms derive from a single fault. Our work can be extended to recognize multiple independent faults in the following way: When the retrieved case fails to explain some of the observed symptoms the system may stand by for additional symptoms, or search the case library again for an additional match that will explain the remaining symptoms. If a new case is found, its causal explanation will be used as if these symptoms were the result of another fault. Using the model the system will try to establish a relation between the two faults by searching for a causal, structural, or functional link between them. If no link is found it may be assumed that the system is experiencing multiple faults. In this situation the reasoner may retrieve an additional case from the library, one that would explain the additional symptoms.

6.3.4 Simulation and the Physical Model

We have indicated that our methodology uses a physical dependency digraph as one of its models. This is a makeshift measure, however, due to the fact that physical fault propagation, being the result of catastrophic component failures, is highly unpredictable. One expedient for dealing with this unpredicatability is to refer to previous cases, as Epaion does; another is to utilize spatial simulation models (SSMs) to determine the effect of uncontrolled energy releases. [Feyock & Li, 1990, 1992] describe the use of SSMs to predict both fluidic and energy leaks. These models, which are easily interfaced with host systems, require only the identity of the faulty component, which can be supplied by Epaion. The SSM then looks in the component database to determine the location and type of the component. If the component is of a type that can cause a fluid or energy leak, the system uses this information to set the initial conditions for the simulation. The simulation is then run, and the physical propagation paths predicted by the SSM are extracted from the run data.

In addition to addressing the chaotic nature of physical propagation, the use of simulation models in conjunction with more traditional reasoning systems is prompted by a belief that deriving answers to real-world questions by setting up the initial conditions of simulation models, running the simulations, and extracting information from the results of the run, constitutes a powerful but underutilized mode of operation for AI systems.

6.4 Conclusion

We have described a case-based reasoning methodology for fault diagnosis and prognosis of physical systems in operation. A hybrid reasoning process based on a library of previous cases and a model of the physical system is used as basis for the reasoning process. This arrangement provides the methodology with the flexibility and power of first-principle reasoners, coupled with the speed of associational systems. Although domain independent, this work is being tested in the domain of aircraft systems fault dia- and prognosis with very promising initial results.

A major concern of this project has been to create a system capable of achieving a practically useful level of performance on a case base of significant size, thereby avoiding the "toy problem" trap besetting many AI systems. The extensive use of a classification hierarchy allows the system to achieve O(log n) search times, while the information abstraction attendant on accident reconstruction produces space-efficient representations. The system is currently hosted on a desktop personal computer, and is estimated to be capable of storing the full set of propulsion-related aircraft accident for the last 20 years. These considerations, together with the encouraging level of success achieved by Epaion, support the expectation that this system will prove to be an effective contributor to aircraft safety.

Appendix A

LIMAP: A modeling tool

A.1 Introduction

A *representation* is a set of syntactic and semantic conventions that make it possible to describe things. Experience has shown that designing a good representation is often the key to turning hard problems into simple ones. According to [Winston 1984] good representations:

- Make important things explicit
- Expose natural constraints, facilitating some class of computations
- Are complete and concise
- Facilitate computation. We can store and retrieve information rapidly.
- Suppress detail.
- Are computable by an existing procedure.

All representations must provide some way to denote objects and to describe the relations that hold among them. Consequently, many representations are built around some form of *semantic net*, since semantic nets denote objects and describe relations among them.

Most AI search/representation techniques are oriented toward a potentially infinite domain of objects and arbitrary relations among them. Experience has shown that in reality much of what needs to be represented in AI can be expressed using a finite domain and unary or binary predicates. Unary predicates can describe object attributes and binary predicates describe relations among two objects.

Well-known vector- and matrix-based representations are appropriate for finite domains and unary/binary predicates, since they satisfy the above-mentioned properties of a "good" representation, and allow the extraction of path information by generalized transitive closure/path matrix computations. In this scheme vectors are used for unary relations and matrices for binary relations. Unfortunately as the number of objects increases the size of matrices rapidly surpasses the amount of available memory in most machines.

Overcoming memory limitations raises the need for abstract data types to represent sparse matrices. These are well suited for most applications, since semantic nets usually represent a limited number of connections among objects, even when working in large domains.

A.2 Matrices and Semantic Nets

A directed graph (digraph) is a 2-tuple <N, E>, where N is a finite set of nodes, and E a finite set of edges. An edge is a member <a, b> of $N \times N$. A labeled digraph is a 3-tuple <N, E, L>, where N is as before, L is a finite set of labels, and E is a finite set of labeled edges, with labels in L. A labeled edge (with label in L) <a, l, b> is a member of $N \times L \times N$.

It is easy to see that digraphs are a graphic representation of binary predicates over finite domains. If P(x, y) is a predicate over domain $D \times D$, then digraph G = <N, E> represents P if P(a, b) iff <a, b> $\in$ E.

Whereas an unlabeled digraph can represent a single predicate, labeled digraphs whose label set is a set of predicate names can represent multiple binary predicates over the same domain $D \times D$ simply by letting edge <a, p, b> denote the fact that predicate p(a, b) is true; the absence of such an edge denotes that p(a, b) is false. Extending the notation, we allow edges to be labeled with *sets* of predicate names; an edge <a, {p_1, ..., p_n}, b> is an abbreviation for the set of edges <a, p_1, b>, ..., <a, p_n, b>. Labeled digraphs thus correspond to the familiar semantic net construct of AI.

Given the problem of representing a unary predicate P(x) over a finite domain D of fixed size n, an obvious and familiar solution is to use boolean vectors, a.k.a. bit strips: for any di in D, P(d_i) is true (false) iff the i'th component of the vector representing P is a 1 (0). Boolean operations such as AND, OR, and NOT on predicates over D are then representable by the corresponding operations over bit strips, which are efficient on most computers. Similarly, binary predicates Q(x, y) over D × D can be efficiently represented by N × N matrices whose ij element is 1 if Q(d_i, d_j) is true, else 0.

Boolean matrices can in principle represent labeled digraphs: a separate matrix is assigned to each label, and represents the subgraph of nodes connected by edges bearing that label. In practice this representation can become unwieldy. The number of different labels may be large, resulting in proliferation of adjacency matrices. Moreover, queries such as ``is there any path (regardless of labels) from node a to node b?" require that the matrices for all labels be ORed together. An answer to the follow-up query ``what are these paths?" is even more difficult to generate from this representation. Such considerations motivate the adoption of symbolic matrices as representation for labeled digraphs. Element ij of a symbolic matrix is P iff the arrow from d_i to d_j in the semantic net has label P, else NIL.

A.3 Implementation

LIMAP (LIsp-based MAtrix Processor) is a set of Common LISP [Steele 1984] procedures that define and manipulate a vector/matrix-based knowledge representation. The user may represent relations among objects using boolean or symbolic matrices/vectors. These matrices/vectors are abstract data types that can represent data (boolean values or symbols) stored in arrays. The semantic interpretation of this data is left to the user.

As is the case for a traditional database system, LIMAP's capabilities are invoked via a language interface that consists of two parts. One is the data definition language (DDL) for specifying both the data the system is to contain as well as "metadata;" i.e. information about the structure and constraints that govern the data contained in the system. The other is the data manipulation language (DML), the subset of the language concerned with the specification of queries and updates on the data. We will categorize the LIMAP functions accordingly. As in Common LISP, LIMAP's functions and arguments are not case sensitive.

A.3.1 DDL Operations

Figure A.1 shows LIMAP's data definition procedures, and their associated syntax.

```
DEFREL <relname> <specs> <type> <rep>
DELREL <relname>
<relname> ::= <symbolic atom>
<type> ::= Bmatrix | Smatrix | Bvector | Svector
<specs> ::= (<length> <length>) | (<length>)
<length> ::= <symbolic atom> | <integer>
<rep> ::= vector-rep | array-rep | sparse-rep
```

Figure 6.1: LIMAP's data definition procedures, and their associated syntax

DEFREL

The function *defrel* defines a relationship with name *relname* of type *type* and having particular specifications. The actual data of the relation is stored in a system-generated variable and is represented according to the *rep* field. Valid representations are array, vector, and sparse representations. The representation is transparent to the user since s/he views all relationships according to their type attribute. Valid *type* attributes are boolean matrices/vectors (Bmatrix/Bvector) and symbolic matrices/vectors (Smatrix/Svector). The *specs* field specifies the dimensions of the matrices/vectors. Matrices are two-dimensional and vectors one-dimensional. When assigning the dimensions of a relation the system expects a list with one or two numbers or symbols. For a matrix definition the first number specifies the number of rows and the second the number of columns. If a symbol is specified the system expects that the symbol is the name of a set of values and substitutes the size of the set for the symbol. Following the definition a change in the size of the set does not affect the dimensionality of the matrix/vector. Change of dimensionality is achieved via the RESIZE function, as is explained later. The following example defines a matrix named "example_mtrx" to be of boolean type, have 4x4 elements represented as a list, a vector named "is_sensor" to be of type boolean and have size of nine elements, and a matrix named "engine" to be a symbolic matrix of size 9x9 and be represented as an array.

```
(setvar '*comps* '(fan compressor combustor turb1 turb2 N1 N2 EGT EPR))
(defrel 'example_mtrx '(4 4) 'Bmatrix 'sparse-rep)
(defrel 'is_sensor '(*comps*) 'Bvector 'vector-rep)
(defrel 'engine '(9 *comps*) 'Smatrix 'array-rep)
```

Matrices/vectors of boolean type are matrices/vectors where each of the elements is either a "0" or "1", representing false or true respectively. The elements of matrices/vectors which are declared as *symbolic* can contain arbitrary s-expressions such as numbers, symbols, or lists. When a matrix is declared as having *array-rep* representation the matrix is associated with a Common LISP two-dimensional array, and every operation on the matrix (such as a retrieval, multiplication etc) is performed on the two-dimensional array. A vector with a *vector-rep* representation is represented as a one-dimensional array. Matrices/vectors having a *sparse-rep* representation are represented as LISP lists. For example, if example_mtrx has only two elements, say a value 1 in row 10, column 30 and in row 33 column 90, then it is represented by the list ((10 30 1) (33 90 1)).

When a relation is defined using *defrel* it is placed in a system-maintained definition table which maintains information about all the defined relations. Additionally, a system-generated variable is bound to the data structure that will actually hold the data. This data structure is a list initialized to nil if the representation is *sparse-rep*. When the representation is *array-rep* the data structure is an array initialized with zeros or nil, depending on the type field. Matrices/vectors of boolean type are initialized to all zeros, and symbolic matrices/vectors to all nils. Figure A.2 shows the contents of the definition table after the above definitions. The *path* and *flag* fields are explained in a later section during the description of the *paths* operation.

NAME	SPECS	TYPE	REP	PATH	FLAG
example_mtrx	(4 4)	Bmatrix	sparse-rep	P0	T
is_sensor	(*comps*)	Bvector	vector-rep	None	None
engine	(9 *comps*)	Smatrix	array-rep	P1	T

Figure 6.2: Contents of the Definition Table

DELREL

The operation *Delrel* deletes a relation by extracting it from the definition table and disassociating all data variables with the relation.

DML Operations

Using the data manipulation language (DML) procedures the user may query a relation's type, specifications, representation, or the actual data stored. S/he may store/retrieve values to particular locations, multiply two matrices, copy one matrix to another, invert, transpose, resize, clear, or take the transitive closure of a matrix. Figure A.3 tabulates the DML operations.

STORE and RETRIEVE

The function STORE allows the user to store a specific value to a particular location in some matrix/vector. The user must specify the matrix/vector name, the value to be stored and the coordinates of the location in row-major order. The function returns the stored value. An error is returned when there is a type mismatch between the value to be stored and the type of the matrix/vector. RETRIEVE retrieves the contents of a particular location in a matrix/vector. In the event that that the specified matrix/vector in not defined, both functions display an appropriate message and return nil.

Example: (STORE 'engine 'N 0 5) stores the symbol "N" in location (0 5) of the symbolic matrix "engine", and returns "N".

TYPE and REP

The functions TYPE and REP notify the user about the type and representation scheme of a particular matrix. The possible return values of *Type* are Bmatrix, Bvector, Smatrix and Svector. REP returns either array-rep, *vector-rep* or *sparse-rep* depending on the specified matrix/vector.

Predicate	Arg1	Arg2	Arg3	Arg4	Arg5	Description
DEFTABLE	[:ALL]	[T]				Display definition table
DISPLAY	relname					Display a relation
TYPE	relname					Relation's type
DIMS	relname					Relation's dimensions
REP	relname					Relation's representation
DATA	relname					Relation's data
DATA-NAME	relname					Relation's data variable
STORE	relname	value	[row]	column		Store value
RETRIEVE	relname	[row]	column			Retrieve contents
TCLOSE	relname					Transitive closure
PATHS	relname	row	column	[:NAME]	[T]	All paths
MULT	relname1	relname2	relname3			Multiply
TRANSPOSE	relname1	relname2				Transpose
CLEAR	relname					Initializes a relation
COMPLEMENT	relname1	relname2				Relation's complement
RESIZE	relname	specs				Changes the dimensions
COPYREL	relname1	relname2				Copy rel1 into rel2
COLUMN	column	relname	vectname			Copy a column into a vector
ROW	row	relname	vectname			Copy a row into a vector
RELAND	relname1	relname2	relname3			Logical AND
RELOR	relname1	relname2	relname3			Logical OR

Figure 6.3: LIMAPs DML operations

Example: (TYPE 'engine) and (REP 'engine) return *Smatrix* and *array-rep* respectively.

DIMS, DATA and DATA-NAME

The function DIMS returns a list of the dimensions of a specified matrix/vector. This list contains one number, the number of elements, for a vector and two numbers, the number of rows and the number of columns, when the specified structure is a matrix. Functions DATA and DATA-NAME, respectively, return the data structure and the symbolic name of the data structure that holds the actual data of the specified abstracted matrix/vector. All of the above functions terminate gracefully with an appropriate message when the specified matrix/vector is not defined.

Example: (DIMS 'engine) returns (9 9) and (DIMS 'engin) returns nil and displays the message "*** From LIMAP, *engin* is not defined".

(DATA-NAME 'engine) returns "engine", while (DATA 'engine) returns

```
nil  N    nil  nil  N    N    nil  nil  nil
nil  nil  N    N    nil  nil  N    nil  nil
nil  nil  nil  N    nil  nil  nil  nil  nil
nil  N    nil  nil  N    nil  nil  nil  nil
N    nil  nil  nil  nil  nil  nil  N    N
nil  nil  nil  nil  nil  nil  nil  nil  nil
nil  nil  nil  nil  nil  nil  nil  nil  nil
nil  nil  nil  nil  nil  nil  nil  nil  nil
nil  nil  nil  nil  nil  nil  nil  nil  nil
```

MULT

MULT is a function that allows the user to multiply two matrices,, a matrix and a vector or two vectors of the same type. The resulting matrix/vector is placed in a user-specified matrix/vector, which constitutes the third argument in the function. If the specified matrices/vectors are not defined, have incompatible types, or incompatible dimensions the function terminates gracefully by displaying appropriate error messages. MULT operates on the following principle.

For boolean matrices/vectors such as b1 an m × n, and b2 an n × r (MULT 'b1 'b2 'b3):

$$b_3[i,j] = \bigvee_{h=1}^{n} f(b_1[i,h], b_2[h,i])$$

where f(x,y) = 1 if both x,y are 1, else 0

For symbolic matrices/vectors such as s1 an m × n, and s2 an n × r (MULT 's1 's2 's3):

$$s3[i,j] = \bigvee_{h=1}^{n} f(s1[i,h], s2[h,i])$$

where f(x,y) = t if both x,y are non-nil, else nil

Examples: The following shows the contents of *example_mtrx* before and after the operation (MULT 'example_mtrx 'example_mtrx 'example_mtrx).

```
Before:                          After:
          1   1   1   1                    1   1   1   1
          1   .   .   .                    1   1   1   1
          1   .   .   .                    1   1   1   1
          .   .   .   .                    .   .   .   .
```

Assuming that S1, S2, and S3 are 4 × 4 symbolic matrices, the following shows the contents of S1, S2, and S3 after the operation (MULT 'S1 'S2 'S3).

```
S1:                   S2:                   S3:
a   nil  nil  nil     b    b    b    b      t   t   t   t
a   nil  nil  nil     nil  nil  nil  nil    t   t   t   t
a   nil  nil  nil     nil  nil  nil  nil    t   t   t   t
a   nil  nil  nil     nil  nil  nil  nil    t   t   t   t
```

TRANSPOSE and COMPLEMENT

The functions TRANSPOSE and COMPLEMENT respectively transpose and complement a specified matrix/vector. TRANSPOSE works only on matrices, and COMPLEMENT inverts zeroes to ones and vice-versa on boolean matrices/vectors. The resulting complemented matrix/vector replaces the specified matrix/vector, but the result of the transposition is placed in a new matrix specified by the user. Successful termination of the above functions returns true.

Examples: The following shows the contents of *example_mtrx* before and after the operation (TRANSPOSE 'example_mtrx 'example_mtrx).

```
Before:                 After:
        1  .  .  .              1  1  1  .
        1  .  .  .              .  .  .  .
        1  .  .  .              .  .  .  .
        .  .  .  .              .  .  .  .
```

The contents of *example_mtrx* before and after the operation (COMPLEMENT 'example_mtrx 'example_mtrx) is as follows:

```
Before:                          After:
            1   .   .   .                  .   1   1   1
            1   .   .   .                  .   1   1   1
            1   .   .   .                  .   1   1   1
            .   .   .   .                  1   1   1   1
```

CLEAR and COPYREL

CLEAR initializes the contents of a specified abstracted matrix/vector. Matrices/vectors of a boolean type are initialized to all zeroes and matrices/vectors of symbolic type to all nils. COPYREL copies one matrix/vector to another. Both arguments must be of the same type and have the same representation. The first argument is the source and the second the destination.

RESIZE

The function RESIZE changes the dimensions of a specified matrix/vector. The first argument is the specified matrix/vector and the second a list containing the new dimensions. After a RESIZE operation that increases the size of the matrix/vector the matrix/vector retains its elements and the newly created locations are initialized with the default values. The newly created locations are appended at the ends of vectors, and the right and bottom margins of matrices. A RESIZE operation that decreases dimension sizes drops higher indexed elements. Thus following a RESIZE operation where the new dimensions are smaller than the previous if the users tries to access locations that don't exist the operation returns nil and prints an out of range error message. Example: If the contents of vector *is_sensor* (defined previously as a boolean vector of size 9) is:

Location: 0 1 2 3 4 5 6 7 8

Contents: 0 0 0 0 0 1 1 1 1

then after the operation (RESIZE 'is_sensor '(11)) the contents of the same vector will be:

Location: 0 1 2 3 4 5 6 7 8 9 10

Contents: 0 0 0 0 0 1 1 1 1 0 0

RELAND and RELOR

The logical functions RELAND and RELOR perform the logical *AND* and *OR* among two vectors/matrices. These vectors/matrices must be of a boolean type and have the same size. The result of either function becomes the contents of the third argument.

Example: The following shows the result of (RELAND 'example_mtrx 'other_mtrx 'result_mtrx), where the contens of *example_mtrx* and *other_mtrx* is:

example mtrx:				**other mtrx:**				**result mtrx:**			
1	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	1	.	.	.	1	.	.	.
1	.	.	.	1	.	.	.	1	.	.	.
.	.	.	.	1	.	.	.	.	.	.	.

When *example_mtrx* and *other_mtrx* have the same contents as above, then following the operation (RELOR 'example_mtrx 'other_mtrx 'result_mtrx) the contents of *result_mtrx* is:

example mtrx:				**other mtrx:**				**result mtrx:**			
1	.	.	.	.	.	.	.	1	.	.	.
1	.	.	.	1	.	.	.	1	.	.	.
1	.	.	.	1	.	.	.	1	.	.	.
.	.	.	.	1	.	.	.	1	.	.	.

DISPLAY, SHOW-ARRAY, and DEFTABLE

LIMAP provides the user with functions that allow him/her to view the contents of matrices/vectors. DISPLAY produces a formatted display of a matrix or a vector. In case the abstracted data types are of symbolic type an "S" is displayed at the location that a symbol exists. In order to see the actual symbols SHOW-ARRAY should be used. The function DEFTABLE displays the contents of the definition table, which contains all the defined matrices/vectors and their associated attributes.

Example: The outputs of (DISPLAY 'enginea) and (SHOW-ARRAY 'enginea) are as follows: (DISPLAY 'enginea)(SHOW-ARRAY 'enginea) .

```
 (DISPLAY 'enginea)                  (SHOW-ARRAY 'enginea)

0:  .  S  .  .  S  S  .  .  .      nil  1    nil  nil  1    1    nil  nil  nil
1:  .  .  S  S  .  .  S  .  .      nil  nil  1    1    nil  nil  1    nil  nil
2:  .  .  .  S  .  .  .  .  .      nil  nil  nil  1    nil  nil  nil  nil  nil
3:  .  S  .  .  S  .  .  .  .      nil  1    nil  nil  1    nil  nil  nil  nil
4:  S  .  .  .  .  .  .  S  S      1    nil  nil  nil  nil  nil  nil  1    1
5:  .  .  .  .  .  .  .  .  .      nil  nil  nil  nil  nil  nil  nil  nil  nil
6:  .  .  .  .  .  .  .  .  .      nil  nil  nil  nil  nil  nil  nil  nil  nil
7:  .  .  .  .  .  .  .  .  .      nil  nil  nil  nil  nil  nil  nil  nil  nil
8:  .  .  .  .  .  .  .  .  .      nil  nil  nil  nil  nil  nil  nil  nil  nil
```

A.3.2 Path Operations: TCLOSE and PATHS

The TCLOSE and PATHS operations form the core of LIMAP's path manipulation capability. The function TCLOSE calculates the transitive closure of a specific matrix. The transitive closure of a matrix M is a matrix M* that contains an entry in location <a, b> iff the directed graph represented by M contains a path (of length 0 or greater) from a to b. In LIMAP M* inherits M's type and representation attributes. Warshall's Algorithm is an efficient method for computing M*, given a matrix M. Intuitively, the algorithm scans the matrix top to bottom, left to right. If an entry is encountered, say in row i, column j, then row i is replaced by row i OR row j, and the scan continues from position ij. Figure 4 shows the code that performs the transitive closure for boolean and symbolic matrices using Warshall's Algorithm.

```
(DEFUN BTclose (rel); Function to compute transitive
    (LET ((max (FIRST (dims rel))); closure of a boolean matrix
        (DO ((k 0 (+ k 1))) ((= k max) nil); Scan top to bottom
            (DO ((i 0 (+ i 1))) ((= i max) nil); Scan left to right
                (COND ((= (retrieve rel i k) 1); If there is an entry
                    (DO ((j 0 (+ j 1))) ((= j max) nil)
                        (store rel (LOGIOR (retrieve rel i j); Swap i and j
                        (retrieve rel k j)) i j))
                    )   ; Close DO
                )   ; Close COND
            )   ; Close DO
        )   ; Close DO
    )   ; Close LET
)   ; Close BTclose
```

```
(DEFUN STclose (rel); Function to compute transitive
    (LET ((max (FIRST (dims rel))); closure of a symbolic matrix
        (DO ((k 0 (+ k 1))) ((= k max) nil); Scan top to bottom
            (DO ((i 0 (+ i 1))) ((= i max) nil); Scan left to right
                (COND ((NOT (NULL (retrieve rel i k))); If there is an entry
                    (DO ((j 0 (+ j 1))) ((= j max) nil)
                        (COND ((NOT (NULL (OR
                            (retrieve rel i j); If there is a symbol in (i, j)
                            (retrieve rel k j)))); OR in (k, j)
                            (store rel t i j))); Then flag that (i,j) are connected
                    )    ; Close DO
                )    ; Close COND
            )    ; Close DO
        )    ; Close DO
    )    ; Close LET
)    ; Close STclose
```

Figure 6.4: Code for transitive closure

Example: Let us assume that we have a network of four nodes labeled as 0, 1, 2, and 3. Assume

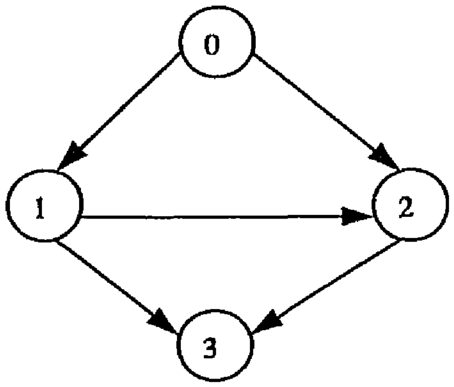

Figure 6.5: Example network

that there are direct connections from 0 to 1, 0 to 2, 1 to 2, 2 to 3, and 1 to 3 as indicated in Figure A.5. The following displays how the network may be represented in a boolean matrix along with the contents of the same matrix after the transitive closure has been computed.

```
Example_mtrx:                   After (TCLOSE
                                'example_mtrx):

        .   1   1   .                   .   1   1   1
        .   .   1   1                   .   .   1   1
        .   .   .   1                   .   .   .   1
        .   .   .   .                   .   .   .   .
```

A value of 1 in locations (0, 1), (0, 2), (1, 2), (1, 3), and (2, 3) means that there is a direct connection between the corresponding nodes. Following the operation of transitive closure a value of 1 in location (i, j) means that there is a connection from node *i* to node *j*. This connection may be direct or indirect. An example of indirect connection is the connection between node 0 and 3. Such a connection is achieved via node 1 or node 2 (Figure A.5).

Assuming that LIMAP's matrices (boolean or symbolic) represent directed graphs, the function PATHS returns all the paths between two specific nodes in a network. Besides the user-defined attributes that characterize each matrix in LIMAP, every matrix is associated with an internal system matrix called the path matrix, and a *flag* field. When PATHS is invoked for the first time on a particular matrix it does the following. First, using an extension of Warshall's Algorithm, all possible paths among every node in the matrix are calculated. A path is a list of node numbers. The resulting lists of paths become the entries of the associated path matrix and the *flag* is set to *nil.* At the end only the paths among the two specific nodes specified by the user are returned. A subsequent request for paths need not recalculate all the paths, but merely retrieve the appropriate entry from the *path matrix.* In case that there is a change in the contents of the user defined matrix (i.e. a change in the graph) the flag field is set to "t" and a subsequent user request for paths triggers a recalculation of the *path matrix.*

In order to operate on a symbolic matrix and produce a path matrix whose ij entry contains the set of all paths from node i to node j, Warshall's algorithm was extended as Figure 6 indicates.

```
Let M be an NxN square matrix
    for k=1 to N ; Scan from top down
     for i=1 to N ; Scan from left to right
        if ( i ≠ k AND M[i,k] ≠ nil ) then
         for j=1 to N
            M[i,j] := UNION ( M[i,j], LINK( M[i,k], M[k,j] ))
```

Figure 6.6: Warshall's Algorithm

In the algorithm of Figure 6, UNION is the normal union operation on sets and LINK operates on lists of paths. If p, q are paths where $p = (v_1, \dots, v_k)$ and $q = (v_k, \dots, v_r)$ then LINK (p,q) returns $(v_1, \dots, v_k, \dots, v_r)$.

More precisely: let E(k) denote the set of all paths going through nodes numbered $\leq$ k only (not including the endpoints, which can be arbitrary). Then the original adjacency matrix represents E(0). More precisely, the original matrix has the list ((i j)) in element ij if there is an arrow from i to j, otherwise nil. (This discussion assumes vertices numbered from 1, although Common Lisp dimensions are actually indexed from 0; in that case, the original matrix would represent E(-1).) Warshall's Algorithm scans the matrix from top to bottom, left to right. The scan of the k'th column computes E(k), as follows. When a non-nil element is encountered in an off-diagonal position in column k, say at ik, that element will be a list of E(k-1)-paths from node i to node k. Consider arbitrary element ij of row i; it contains all E(k-1)-paths (if any) from i to j. Now that we also know the E(k-1)-paths from i to k, we can reach j from i either by the E(k-1) paths in ij, or by going from node i to node k, and then from node k to node j by any path (if any) in element kj. Such paths will, of course, be E(k) paths. Thus we add to the paths already at ij the link of all paths from i to k and all paths from k to j.

Example: Assume that *symbolic_mtrx* is a 4 × 4 symbolic matrix that represents the network of Figure 5. If the contents of *symbolic_mtrx* is as follows, then (PATHS 'example_mtrx 0 1) returns "(0 1)", and (PATHS 'symbolic_mtrx 0 3) returns "((0 1 3) (0 2 3) (0 1 2 3))". The contents of *symbolic_mtrx* and the internally maintained path matrix are:

symbolic_mtrx:

	0	1	2	3
0:	nil	1	1	nil
1:	nil	nil	1	1
2:	nil	nil	nil	1
3:	nil	nil	nil	nil

corresponding path matrix:

	0	1	2	3
0:	nil	((0 1))	((0 2))	((0 1 3) (0 2 3) (0 1 2 3))
1:	nil	nil	((1 2))	((1 3))
2:	nil	nil	nil	((2 3))
3:	nil	nil	nil	nil

Following our exemplar definitions, Figure 7 displays the contents of the definition table including the attributes of each path matrix (compare with figure 2). This is achieved by using the optional *all* flag of the DEFTABLE function, i.e by calling (DEFTABLE :all t). In the path field of the table is stored the name of the associated path matrix. The associated path matrices have no value in the flag field.

	NAME	TYPE	SPECS	REP	PATH	FLAG
0:	example_mtrx	(4 4)	Bmatrix	sparse-rep	P0	T
1:	P0	(4 4)	Bmatrix	sparse-rep	None	None
2:	is_sensor	(*comps*)	Bvector	vector-rep	None	None
3:	engine	(9 *comps*)	Smatrix	array-rep	P1	T
4:	P1	(9 *comps*)	Smatrix	array-rep	None	None

Figure 6.7: Contents of the definition table

A.3.3 Control Structures

Queries of the form "is there a relation R such that nodes a and b are in relation R? "is there a path from x to y? a path fulfilling constraint C? where can I go from x? how can I get to x?" arise frequently both in AI and elsewhere. Such queries, which involve quantification over relations, correspond to statements in the *second-order predicate calculus*. This section describes the control structures that make LIMAP an efficient second-order predicate calculus programming system.

The distinction between procedural and non-procedural predicate calculus specifications blurs if the underlying domain is finite, since the FORALL and EXISTS quantifiers map in an obvious way to loops ranging over the domain elements. It has been our goal to give the LIMAP data manipulation language as non-procedural a character as possible. In particular, LIMAP notation is an adaptation of the (function-less) predicate calculus, with extensions to allow data retrieval in addition to data specification. For example, a "yes" answer to (EXISTS X) (FORALL Y) P(X,Y) is insufficient; the actual X-value must be retrieved. We have found that minimal modifications of the control macros described in [Charniak et al., 1987] were suitable for the task of expressing the required quantifications. Following is a summary of the general form of the control structure implemented by these macros:

```
(FOR (<variable_1> :IN <set_1>)
     . . .
     (<variable_n> :IN <set_n>)
     [:WHEN <when-expression>]
     <FOR-keyword> <expression_1> . . . <expression_n>)
```

The *expression*$_i$ following *FOR-keyword* are called the *body* of the FOR. The construct (<*variablei*> :IN <*set*$_i$>) causes the variable to iterate over the elements of the set, which may be specified as a list, a vector, or a matrix row or column. When there are several sets the FOR interates over the elements of each set in the following way. Initially the first element of each set is assigned to the corresponding *variable*$_i$ and the body of the FOR is evaluated. Then the second element of each set is assigned to the corresponding *variable*$_i$ and the body is evaluated again. This is repeated until some set runs out of elements or the final value of the FOR is determined as governed by the *FOR-keyword.*

FOR-keywords

:ALWAYS	return true if all the values of body are true
:FILTER	produce a list of the non-nil values of body
:FIRST	produce the first non-nil value of body
:SAVE	produce a list of all values of body

While the description of these constructs is procedural in form, the effect when programming in this notation is that of writing FORALLs and EXISTs, with the proviso that any variable values that are found to "EXIST" are collected in accordance with the FOR-keyword and returned as value. The following section contains an example application of LIMAP.

A.4 Conclusion

We have described a programming system oriented toward efficient information representation/ manipulation over fixed finite domains, and quantification over paths and predicates. The initial motivation for the creation of such a system was the fact that the need for such operations arose frequently in the domain of diagnosis/prognosis generation problem domain. Since then it has become apparent that the facilities provided are applicable to problems both within and outside of AI.

Our experience to date has shown that LIMAP is applicable to a wide range of problems. While LIMAP, if abused, is as capable of inefficient operation as any other misused programming system, we have found that for every problem yet attempted there has existed a LIMAP formulation that was concise, comprehensible, and for which LIMAP's facilities constituted an efficient problem representation.

Appendix B

Case Library

B.1 Fuel Metering Unit

This is a propulsion failure scenario staged at Boeing Inc. and reported in [Shontz et. al. 1992]. The failure is a malfunctioning valve in the fuel metering system.

```
.; Fault scenario F1
; Flight Deck Engine Advisor, Boeing document D6-55880, May 1991
;
(setf *case1*
 '(
                                          ; -- Fault
   (fault i-m-fuel-metering-unit)
   (events m-group
       (1 m-causal-event
           (ante i-m-fuel-metering-unit)
           (cnsq i-m-fuel-flow))
       (2 m-causal-event
           (ante i-m-fuel-flow)
           (cnsq i-m-n1))
        (3 m-causal-event
           (ante i-m-n1)
           (cnsq i-m-n2))
        (4 m-causal-event
            (ante i-m-n2)
           (cnsq i-m-egt)))

   (id "Boeing Test Flight F-1")
   (date "1")
   (airline "flight test data")
   (flight "flight test data")
   (aircraft "flight test data")

                                         ; -- Context Variables
   (phase-of-flight i-m-ground-start)
    (weather i-m-clear)
   (workload i-m-moderate)
   (engine-commanded-status i-m-start)
                                         ; -- Symptoms
 (egt m-group
   (1 m-sensor-reading (status i-m-normal) (trend i-m-increases))
   (2 m-sensor-reading (status i-m-normal) (trend i-m-increases))
   (3 m-sensor-reading (status i-m-high) (trend i-m-increases))
   (4 m-sensor-reading (status i-m-high) (trend i-m-increases)))
 (n1 m-group
   (1 m-sensor-reading (status i-m-normal) (trend i-m-increases))
   (2 m-sensor-reading (status i-m-low) (trend i-m-increases))
   (3 m-sensor-reading (status i-m-low) (trend i-m-increases))
   (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (n2 m-group
   (1 m-sensor-reading (status i-m-low) (trend i-m-increases))
   (2 m-sensor-reading (status i-m-low) (trend i-m-increases))
   (3 m-sensor-reading (status i-m-low) (trend i-m-increases))
   (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (fuel-flow m-group
    (1 m-sensor-reading (status i-m-low) (trend i-m-increases))
    (2 m-sensor-reading (status i-m-low) (trend i-m-increases))
    (3 m-sensor-reading (status i-m-low) (trend i-m-increases))
    (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))

)
)
```

B.2 Fuel Boost Pump

This is a propulsion failure scenario staged at Boeing Inc. and reported in [Shontz et. al. 1992]. An engine flameout occurred due to a fuel boost pump failure.

```
(setf *case2*
 '(
                                        ; -- Solution Data
    (fault i-m-fuel-boost-pump)
    (events m-group
    (1 m-causal-event
          (ante i-m-fuel-boost-pump)
          (cnsq i-m-fuel-pressure))
    (2 m-causal-event
           (ante i-m-fuel-pressure)
          (cnsq i-m-fuel-flow))
    (3 m-causal-event
          (ante i-m-fuel-flow)
          (cnsq i-m-n1))
    (4 m-causal-event
          (ante i-m-n1)
           (cnsq i-m-n2)))
                                        ; -- Id features
    (id "Boeing Test Flight F-2")
    (date "2")
    (airline "flight test data")
    (flight "flight test data")
    (aircraft "flight test data")

                                        ; -- Context Variables
    (phase-of-flight i-m-cruise)
     (workload i-m-light)
    (engine-commanded-status i-m-steady)

                                        ; -- Symptoms
     (n1 m-group
          (1 m-sensor-reading (status i-m-normal))
          (2 m-sensor-reading (status i-m-normal))
          (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
          (4 m-sensor-reading (status i-m-low) (trend i-m-decreases)))
    (n2 m-group
           (1 m-sensor-reading (status i-m-normal))
           (2 m-sensor-reading (status i-m-normal))
           (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
           (4 m-sensor-reading (status i-m-low) (trend i-m-decreases)))
    (fuel-flow m-group
          (1 m-sensor-reading (status i-m-normal))
          (2 m-sensor-reading (status i-m-normal))
          (3 m-sensor-reading (status i-m-zero))
          (4 m-sensor-reading (status i-m-zero)))
    (fuel-pressure m-group
           (1 m-sensor-reading (status i-m-low) (trend i-m-decreases))
           (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
           (3 m-sensor-reading (status i-m-zero))
          (4 m-sensor-reading (status i-m-zero)))

)
)
```

B.3 Ice Ingestion

This is a propulsion failure scenario staged at Boeing Inc. and reported in [Shontz et. al. 1992]. The failure was a foreign object damage due to light ice ingestion.

```
; Flight Deck Engine Advisor, Boeing document D6-55880, May 1991
(setf *case3*
 '(
 ; -- Solution Data
 (fault i-m-ice-ingestion)
 (events m-group
 (1 m-causal-event
 (ante i-m-ice-ingestion)
 (cnsq i-m-fan-blade-damage))
 (2 m-causal-event
 (ante i-m-fan-blade-damage)
 (cnsq i-m-fan-rotor-imbalance))
 (3 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-fan-vib))
 (4 m-causal-event
 (ante i-m-fan-vib)
 (cnsq i-m-compressor-vib))
 (5 m-causal-event
 (ante i-m-compressor-vib)
 (cnsq i-m-thrust)))

 (id "Boeing Test Flight F-3")
 (date "3")
 (airline "flight test data")
 (flight "flight test data")
 (aircraft "flight test data")
 ; -- Context Variables
 (phase-of-flight i-m-climb-out)
 (weather i-m-icing)
 (workload i-m-moderate)
 (engine-commanded-status i-m-climb-power)

 ; -- Symptoms
 (fan-vib m-group
 (1 m-sensor-reading (status i-m-high) (trend i-m-increases)))
 (compressor-vib m-group
 (1 m-sensor-reading (status i-m-high) (trend i-m-increases)))
 (thrust m-group
 (1 m-sensor-reading (status i-m-low)))

 (fan-vib-behavior-mop i-m-hhhh)
 (thrust-behavior-mop i-m-hhhh)
 (compressor-vib-behavior-mop i-m-llll)
))
```

B.4 Ice Ingestion

This is a propulsion failure scenario staged at Boeing Inc. and reported in [Shontz et. al. 1992]. The failure was a foreign object damage due to moderate ice ingestion

```
; Fault scenario F4
; Flight Deck Engine Advisor, Boeing document D6-55880, May 1991
;
(setf *case4*
 '(
 (fault i-m-ice-ingestion)
 (events m-group
 (1 m-causal-event
 (ante i-m-ice-ingestion)
 (cnsq i-m-fan-blade-damage))
 (2 m-causal-event
 (ante i-m-fan-blade-damage)
 (cnsq i-m-fan-rotor-imbalance))
 (3 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-fan-vib))
 (4 m-causal-event
 (ante i-m-fan-vib)
 (cnsq i-m-compressor-vib))
 (5 m-causal-event
 (ante i-m-compressor-vib)
 (cnsq i-m-n2))
 (6 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-fuel-flow))
 (7 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-egt))
 (8 m-causal-event
 (ante i-m-fuel-flow)
 (cnsq i-m-thrust)))


                                         ; -- id features
(id "Boeing Test Flight F-4")
                                         ; -- Context Variables
(phase-of-flight i-m-climb-out)
(weather i-m-icing)
(workload i-m-moderate)
(engine-commanded-status i-m-climb-power)
                                         ; -- Symptoms
(compressor-vib m-group
(1 m-sensor-reading (status i-m-high))
(2 m-sensor-reading (status i-m-high)))
(thrust m-group
(1 m-sensor-reading (status i-m-low))
(2 m-sensor-reading (status i-m-low) (trend i-m-increases)))
(fan-vib m-group
(1 m-sensor-reading (status i-m-high))
(2 m-sensor-reading (status i-m-high)))
(fuel-flow m-group
(1 m-sensor-reading (status i-m-low))
(2 m-sensor-reading (status i-m-low)))
(n2 m-group
(1 m-sensor-reading (status i-m-low))
(2 m-sensor-reading (status i-m-normal)))
(egt m-group
(1 m-sensor-reading (status i-m-low))
(2 m-sensor-reading (status i-m-low)))


)
)
```

B.5 Ice Ingestion

This is a propulsion failure scenario staged at Boeing Inc. and reported in [Shontz et. al. 1992]. The failure was a foreign object damage due to heavy ice ingestion.

```
(setf *case5*
 '(
 ; -- Solution Data
; (fault i-m-ice-ingestion)
; (events m-group
; (1 m-causal-event
; (ante i-m-ice-ingestion)
; (cnsq i-m-fan-blade-damage))
; (2 m-causal-event
; (ante i-m-fan-blade-damage)
; (cnsq i-m-fan-rotor-imbalance))
; (3 m-causal-event
; (ante i-m-fan-rotor-imbalance)
; (cnsq i-m-broad-band-vib))
; (4 m-causal-event
; (ante i-m-fan-rotor-imbalance)
; (cnsq i-m-n1))
; (5 m-causal-event
; (ante i-m-fan-rotor-imbalance)
; (cnsq i-m-n2))
; (6 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-fuel-flow))
; (7 m-causal-event
; (ante i-m-fuel-flow)
; (cnsq i-m-thrust))
; (8 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-egt)))

 (id "Boeing Test Flight F-5")
 (date "5")
 (airline "flight test data")
 (flight "flight test data")
 (aircraft "flight test data")
 ; -- Context Variables
 (phase-of-flight i-m-cruise)
 (weather i-m-icing)
 (workload i-m-moderate)
 (engine-commanded-status i-m-climb-power)

 ; -- Symptoms
 (n1 m-group
 (1 m-sensor-reading (status i-m-normal)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-low)))
 (egt m-group
 (1 m-sensor-reading (status i-m-low)))
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-low)))
 (thrust m-group
 (1 m-sensor-reading (status i-m-low)))
 (broad-band-vib m-group
 (1 m-sensor-reading (status i-m-high)))

)
```

B.6 Volcanic Ash Ingestion

This is a propulsion failure scenario staged at Boeing Inc. and reported in [Shontz et. al. 1992]. An engine flameout occurred due to volcanic ash ingestion producing fuel nozzle clogging.

```
; Fault scenario F6
; Flight Deck Engine Advisor, Boeing document D6-55880, May 1991
(setf *case6*
 '(
 ; -- Solution Data
 (fault i-m-volcanic-ash-ingestion)
 (events m-group
 (1 m-causal-event
 (ante i-m-volcanic-ash-ingestion)
 (cnsq i-m-fan-blade-damage))
 (2 m-causal-event
 (ante i-m-fan-blade-damage)
 (cnsq i-m-fan-rotor-imbalance))
 (3 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-n1))
 (4 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-n2))
 (5 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-fuel-flow))
 (6 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-egt)))

 (id "Boeing Test Flight F-6")
 (date "6")
 (airline "flight test data")
 (flight "flight test data")
 (aircraft "flight test data")
 ; -- Context Variables
 (phase-of-flight i-m-descent)
 (weather i-m-cloudy)
 (workload i-m-moderate)
 (engine-commanded-status i-m-max-power)

 ; -- Symptoms

 (ash-cloud i-m-visible)
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (2 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (3 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (4 m-sensor-reading (status i-m-high) (trend i-m-increases)))
 (n1 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-decreases)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-decreases)))
 (egt m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-normal))
 (3 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (4 m-sensor-reading (status i-m-high) (trend i-m-increases)))

)
)
```

B.7 Foreign Object Ingestion

On November 12, 1975, an Overseas National Airways DC-10-30 (Flight 32) crashed while attempting to take off from John F. Kennedy International Airport, Jamaica, New York [NTSB-AAR-76-19]. During the m takeoff roll a large number of sea gulls rose from the runway and were ingested into the engine. The number 3 engine disintegrated. The takeoff was rejected and the aircraft crashed off the end of the runway. The NTSB determined that the probable cause of the accident was the disintegration and subsequent fire in the number 3 engine when it ingested a large number of sea gulls.

```
; Overseas National Airways
; NTSB-AAR-76-19
(setf *case11*
 '(
 ; -- Solution Data
 (fault i-m-bird-ingestion)
 (events m-group
 (1 m-causal-event
 (ante i-m-bird-ingestion)
 (cnsq i-m-fan-blade-damage))
 (2 m-causal-event
 (ante i-m-fan-blade-damage)
 (cnsq i-m-fan-rotor-imbalance))
 (3 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-n1))
 (4 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-n2))
 (5 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-fuel-flow))
 (6 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-egt))
 (7 m-causal-event
 (ante i-m-fuel-flow)
 (cnsq i-m-epr)))
 ; -- id features
 (id "Overseas National Airways F-32")
 (date "November 12, 1975")
 (airline "Overseas National Airways")
 (flight "Flight 32")
 (aircraft "DC-10-30")
 ; -- Context Variables
 (phase-of-flight i-m-take-off)
 ; -- Symptoms
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-fluctuates))
 (3 m-sensor-reading (status i-m-fluctuates))
 (4 m-sensor-reading (status i-m-low)))
 (n1 m-group
 (1 m-sensor-reading (status i-m-fluctuates))
 (2 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-high) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-high) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (egt m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (4 m-sensor-reading (status i-m-high) (trend i-m-stable)))
 (epr m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-high) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))

)
)
```

B.8 Fan Blade Damage

On January 8, 1989, a British Midland Airways Boeing 737-400 (G-OBME) was climbing through 28,300 feet when the outer panel of one blade in the fan of the No 1 (left) engine detached. This gave rise to a series of compressor stalls in the No 1 engine, which resulted in airframe shuddering. Believing that the No 2 engine had suffered damage, the crew shut that engine down. The shuddering caused by the surging of the No 1 engine ceased as soon as the No 2 engine was throttled back, which persuaded the crew that they had dealt correctly with the emergency. The aircraft struck a field.

```
; Midlands Accident G-OBME
; Air Accident Report 4/90, Department of Transport
(setf *case12*
 '(
 ; -- Solution Data
 (fault i-m-fan-blade-damage)
 (events m-group
 (1 m-causal-event
 (ante i-m-fan-blade-damage)
 (cnsq i-m-fan-rotor-imbalance))
 (2 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-fan-vib))
 (3 m-causal-event
 (ante i-m-fan-vib)
 (cnsq i-m-n1))
 (4 m-causal-event
 (ante i-m-fan-vib)
 (cnsq i-m-n2))
 (5 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-fuel-flow))
 (6 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-egt)))
 ; -- id features
 (id "Midlands Airways 1989 G-OBME")
 (date "January 1989")
 (airline "Midlands Airways")
 (aircraft "Boeing 737-400")
 (engine "General Electric CFM56")
 ; -- Context Variables
 (phase-of-flight i-m-climb-out)
 ; -- Symptoms
 (fan-vib m-group
 (1 m-sensor-reading (status i-m-high))
 (2 m-sensor-reading (status i-m-high))
 (3 m-sensor-reading (status i-m-high))
 (4 m-sensor-reading (status i-m-high)))
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (n1 m-group
 (1 m-sensor-reading (status i-m-fluctuates))
 (2 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-high) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-high) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (egt m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high))
 (3 m-sensor-reading (status i-m-high))
 (4 m-sensor-reading (status i-m-high)))

)
)
```

B.9 Fan Blade Damage

On June 9, 1989, a Dan Air Boeing 737-400 (G-BNNL) suffered a failure in the number 1 engine [AAIB-AAR-4/90]. The crew identified the failed engine correctly and completed a full shutdown drill. Examination of the engine after landing showed that the fan had been massively damaged, with severe damage to the leading edges of all blades. One blade had fractured close to the root and another just below the midspan shroud, both entirely by overload rupture. There was a third blade fracture, however, just above the mid-span shroud which appeared to be very similar to a blade from number 1 engine of ME.

```
; Dan Air G-BNNL
; Air Accident Report 4/90, Department of Transport
(setf *case13*
 '(
 ; -- Solution Data
 (fault i-m-fan-blade-damage)
 (events m-group
 (1 m-causal-event
 (ante i-m-fan-blade-damage)
 (cnsq i-m-fan-rotor-imbalance))
 (2 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-fan-vib))
 (3 m-causal-event
 (ante i-m-fan-vib)
 (cnsq i-m-n1))
 (4 m-causal-event
 (ante i-m-fan-vib)
 (cnsq i-m-n2))

 (5 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-fuel-flow))
 (6 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-egt)))
 ; -- id features
 (id "Dan Air 1989 G-BNNL")
 (date "June 9, 1989")
 (airline "Dan Air")
 (aircraft "Boeing 737-400")
 (engine "General Electric CFM56")
 ; -- Context Variables
 (phase-of-flight i-m-climb-out)
 (engine-commanded-status i-m-climb-power)
 ; -- Symptoms
 (fan-vib m-group
 (1 m-sensor-reading (status i-m-high))
 (2 m-sensor-reading (status i-m-high))
 (3 m-sensor-reading (status i-m-high))
 (4 m-sensor-reading (status i-m-high)))
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (n1 m-group ; n
 (1 m-sensor-reading (status i-m-fluctuates))
 (2 m-sensor-reading (status i-m-low) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (n2 m-group ; n
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (egt m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high))
 (3 m-sensor-reading (status i-m-high))
 (4 m-sensor-reading (status i-m-high)))

)
)
```

B.10 Fan Blade Damage

On June 11, 1989, a British Midland Airways Boeing 737-400 (G-OBMG) suffered a failure in the number 2 engine [AAIB-AAR-4/90]. The aircraft landed successfully. Examination of the number 2 engine revealed that the outer panel of one fan blade had detached outboard of the mid-span shroud and become lodged in the space between the fan and fan outlet guide vanes. Some damage had occurred to the fan abradable liner and the forward acoustic panels, but apart from this there appeared to have been very little damage to the engine. It was also found that there had been loosening of several pipe unions and of the MEC to fuel pump attachment nuts.

```
; Midland Airways G-OBMG
; Air Accident Report 4/90, Department of Transport
(setf *case14*
 '(
 ; -- Solution Data
; (fault i-m-fan-blade-damage)
; (events m-group
; (1 m-causal-event
; (ante i-m-fan-blade-damage)
; (cnsq i-m-fan-rotor-imbalance))
; (2 m-causal-event
; (ante i-m-fan-rotor-imbalance)
; (cnsq i-m-fan-vib))
; (3 m-causal-event
; (ante i-m-fan-vib)
; (cnsq i-m-n1))
; (4 m-causal-event
; (ante i-m-fan-vib)
; (cnsq i-m-n2))
; (5 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-fuel-flow))
; (6 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-egt)))
 ; -- id features
 (id "Midland Airways 1989 G-OBMG")
 (date "June 11, 1989")
 (airline "Midland Airways")
 (aircraft "Boeing 737-400")
 (engine "General Electric CFM56")
 ; -- Context Variables
 (phase-of-flight i-m-climb-out)
 (engine-commanded-status i-m-climb-power)
 ; -- Symptoms
 (fan-vib m-group
 (1 m-sensor-reading (status i-m-high))
 (2 m-sensor-reading (status i-m-high))
 (3 m-sensor-reading (status i-m-high))
 (4 m-sensor-reading (status i-m-high)))
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (n1 m-group
 (1 m-sensor-reading (status i-m-fluctuates))
 (2 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high) (trend i-m-fluctuates))
 (3 m-sensor-reading (status i-m-high) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (egt m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high))
 (3 m-sensor-reading (status i-m-high))
 (4 m-sensor-reading (status i-m-high)))

)
)
```

B.11 Turbine Blade Separation

On July 19, 1970, a United Airlines' Boeing 737-222 (Flight 611) crashed shortly after taking off from the Philadelphia International Airport [NTSB-AAR-72-9]. During takeoff, the number 1 engine failed. The captain thought that both engines were spooling down and reasoned that they both had failed. Therefore, he decided to reject the takeoff and land the aircraft on the existing runway. The aircraft came to a stop past the end of the runway. The NTSB determined that a first stage turbine blade had failed in the number 1 engine which caused the engine to cease rotation. The number 2 engine was operable throughout the flight.

```
; United Airlines 1970
; Air Accident Report 4/90, Department of Transport
; (setf *case15*
'(
; -- Solution Data
(fault i-m-turbine-blade-separation)
(events m-group
(1 m-causal-event
(ante i-m-turbine-blade-separation)
(cnsq i-m-turbine-rotor-imbalance))
(2 m-causal-event
(ante i-m-turbine-rotor-imbalance)
(cnsq i-m-n1))
(3 m-causal-event
(ante i-m-turbine-rotor-imbalance)
(cnsq i-m-n2))
(4 m-causal-event
(ante i-m-n2)
(cnsq i-m-fuel-flow))
(5 m-causal-event
(ante i-m-fuel-flow)
(cnsq i-m-epr))
(6 m-causal-event
(ante i-m-n2)
(cnsq i-m-egt)))
; -- id features
(id "United Airlines F-611")
(date "July 19, 1970")
(airline "United Airlines")
(aircraft "Boeing 737-222")
(flight "611")
; -- Context Variables
(phase-of-flight i-m-climb-out)
(engine-commanded-status i-m-climb-power)
; -- Symptoms
(epr m-group
(1 m-sensor-reading (status i-m-low) (trend i-m-decreases))
(2 m-sensor-reading (status i-m-low))
(3 m-sensor-reading (status i-m-low))
(4 m-sensor-reading (status i-m-low)))
(fuel-flow m-group
(1 m-sensor-reading (status i-m-normal))
(2 m-sensor-reading (status i-m-low) (trend i-m-decreases)))
(n1 m-group
(1 m-sensor-reading (status i-m-low) (trend i-m-decreases)))
(n2 m-group
(1 m-sensor-reading (status i-m-low) (trend i-m-decreases))
(2 m-sensor-reading (status i-m-zero))
(3 m-sensor-reading (status i-m-zero))
(4 m-sensor-reading (status i-m-zero)))
(egt m-group
(1 m-sensor-reading (status i-m-normal))
(2 m-sensor-reading (status i-m-high) (trend i-m-increases))
(3 m-sensor-reading (status i-m-high) (trend i-m-stable))
(4 m-sensor-reading (status i-m-high) (trend i-m-decreases)))

)
)
```

B.12 Turbine Blade Separation

On May 21, 1978 an American Airlines' Boeing 727 (Flight 566) experienced a engine failure in its number one engine just after rotation on take-off from Greater Cincinnati airport, Cincinnati, Ohio [Schutte]. The captain performed emergency shut-down procedures on the engine and returned to the airport. The National Transportation Safety Board determined that the engine failure was caused by several turbine blade separations. Just after rotation the captain noted that EGT was increasing and then started decreasing.

```
; American Airlines 1978
;
(setf *case16*
'(
 ; -- Solution Data
; (fault i-m-turbine-blade-separation)
; (events m-group
; (1 m-causal-event
; (ante i-m-turbine-blade-separation)
; (cnsq i-m-turbine-rotor-imbalance))
; (2 m-causal-event
; (ante i-m-turbine-rotor-imbalance)
; (cnsq i-m-n1))
; (3 m-causal-event
; (ante i-m-turbine-rotor-imbalance)
; (cnsq i-m-n2))
; (4 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-fuel-flow))
; (5 m-causal-event
; (ante i-m-fuel-flow)
; (cnsq i-m-epr))
; (6 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-egt)))

 ; -- id features
 (id "American Airlines F-566")
 (date "May 21, 1978")
 (airline "American Airlines")
 (aircraft "Boeing 727")
 (flight "566")
 ; -- Context Variables
 (phase-of-flight i-m-take-off)
 (engine-commanded-status i-m-take-off)
 ; -- Symptoms
 (epr m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low)))
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-normal))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low)))
 (n1 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-decreases)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-decreases)))
 (egt m-group
 (1 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (2 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (3 m-sensor-reading (status i-m-high) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-decreases)))

)
)
```

B.13 Engine Separation

On May 25, 1979 an American Airlines DC-10-10 (Flight 191) crashed into an open field northwest of Chicago-O'Hare International Airport [NTSB-AAR-79-17]. During takeoff rotation, the left engine and pylon assembly, and about 3 feet of the leading edge of the left wing separated from the aircraft. The aircraft began to roll to the left until the wings were past the vertical position. During the roll, the aircraft's nose pitched down below the horizon and crashed. The NTSB determined that the probable cause of this accident was the asymmetrical stall and the ensuing roll of the the aircraft at a critical point during takeoff. This was caused by the uncommanded retraction of the left wing outboard leading edge slats and the loss of the stall warning and slat disagreement indication systems resulting from separation of the number 1 engine and pylon assembly. The NTSB determined that the accident would have been survivable had the flight crew known that the stall warning and the slat disagreement indication systems were inoperative.

```
; American Airlines 1979
; Engine Separation
(setf *case17*
 '(
 ; -- Solution Data
 (fault i-m-engine-separation)
 (events m-group
 (1 m-causal-event
 (ante i-m-engine-separation)
 (cnsq i-m-egt))
 (2 m-causal-event
 (ante i-m-engine-separation)
 (cnsq i-m-n1))
 (3 m-causal-event
 (ante i-m-engine-separation)
 (cnsq i-m-n2))
 (4 m-causal-event
 (ante i-m-engine-separation)
 (cnsq i-m-fuel-flow))
 (5 m-causal-event
 (ante i-m-engine-separation)
 (cnsq i-m-epr)))

 ; -- id features
 (id "American Airlines F-191")
 (date "May 25, 1979")
 (airline "American Airlines")
 (aircraft "DC-10-10")
 (flight "191")
 ; -- Context Variables
 (phase-of-flight i-m-take-off)
 (engine-commanded-status i-m-take-off)
 ; -- Symptoms
 (epr m-group
 (1 m-sensor-reading (status i-m-zero)))
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-zero)))
 (n1 m-group
 (1 m-sensor-reading (status i-m-zero)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-zero)))
 (egt m-group
 (1 m-sensor-reading (status i-m-zero)))

)
)
```

B.14 Bad Fuel Controller

On February 19, 1985 a China Airlines' Boeing 747 (Flight 006) was cruising on autopilot when the crew diagnosed a flame-out in the number four engine [NTSB-AAR-86-03]. The engine, had a bad fuel controller and had not flamed out but was suffering from a condition known as "bleed-air hogging." The crew became preoccupied with the failure and did not notice that the controls on the autopilot had reached their maximum allowable correction and that the aircraft was engaged in a right bank. When the captain tried to correct for the problem, he reasoned that all of the attitude indicators had failed (they had not). The ensuing actions put the aircraft into a vertical dive. During the dive the crew made a third mis-diagnosis that all of the engines were flamed-out (in fact, only one engine had flamed out and it is this engine that is used in this case). Finally the captain regained control of the aircraft and all engines were "restarted." The aircraft suffered severe stress damage and made a safe landing in San Francisco. The NTSB determined that the accident was caused by a faulty fuel controller and the flight crew's poor monitoring of systems.

```
; China Airlines 006
; NTSB-AAR-86-03
(setf *case18*
 '(

 ; -- Fault
; (fault i-m-fuel-controller)
; (events m-group
; (1 m-causal-event
; (ante i-m-fuel-controller)
; (cnsq i-m-fuel-flow))
; (2 m-causal-event
; (ante i-m-fuel-flow)
; (cnsq i-m-n1))
; (3 m-causal-event
; (ante i-m-n1)
; (cnsq i-m-n2))
; (4 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-egt))
; (5 m-causal-event
; (ante i-m-fuel-flow)
; (cnsq i-m-epr)))

 (id "China Air F-006")
 (date "February 19")
 (airline "China Air")
 (flight "006")
 (aircraft "Boeing 747")

 ; -- Context Variables
 (phase-of-flight i-m-cruise)
 (workload i-m-high)
 (engine-commanded-status i-m-mid-power)
 ; -- Symptoms
 (egt m-group
 (1 m-sensor-reading (status i-m-normal) (trend i-m-decreases))
 (2 m-sensor-reading (status i-m-low) (trend i-m-stable))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (n1 m-group
 (1 m-sensor-reading (status i-m-normal) (trend i-m-decreases))
 (2 m-sensor-reading (status i-m-low) (trend i-m-stable))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-normal) (trend i-m-decreases))
 (2 m-sensor-reading (status i-m-low) (trend i-m-stable))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (epr m-group
 (1 m-sensor-reading (status i-m-normal) (trend i-m-decreases))
 (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (2 m-sensor-reading (status i-m-low) (trend i-m-stable))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))

)
)
```

B.15 Volcanic Ash Ingestion

On December 15, 1989 a Boeing 747-400 was flying at 25,000 feet near Anchorage, Alaska when it experienced flameouts on all four engines [Lloyd 1990]. The flameouts were due to volcanic ash ingestion from a cloud produced by an eruption of Mt. Redoubt during the previous day. The flight crew restarted engines No 1 and 2 at 13,000 feet and were able to maintain altitude as they restarted the remaining engines. The airplane made an uneventful landing at Anchorage.

```
; Mount Redoubt, 1989
(setf *case19*
 '(
 ; -- Solution Data
 (fault i-m-volcanic-ash-ingestion)
 (events m-group
 (1 m-causal-event
 (ante i-m-volcanic-ash-ingestion)
 (cnsq i-m-fan-blade-damage))
 (2 m-causal-event
       (ante i-m-fan-blade-damage)
       (cnsq i-m-fan-rotor-imbalance))
 (3 m-causal-event
       (ante i-m-fan-rotor-imbalance)
       (cnsq i-m-n1))
 (4 m-causal-event
       (ante i-m-fan-rotor-imbalance)
       (cnsq i-m-n2))
 (5 m-causal-event
 (ante i-m-n2)
 (cnsq i-m-egt)))


 (id "Mount Redoubt")
 (date "December 15, 1989")
 (aircraft "Boeing 747-400")
 (engine "General Electric CF6-80-C2")
 ; -- Context Variables
 (phase-of-flight i-m-descent)
 (weather i-m-cloudy)
 ; -- Symptoms

 (glow-in-engines i-m-visible)
 (smoke i-m-visible)
 (n1 m-group
 (1 m-sensor-reading (status i-m-low))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-low))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (egt m-group
 (1 m-sensor-reading (status i-m-low))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (epr m-group
 (1 m-sensor-reading (status i-m-low))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))

)
)
```

B.16 Volcanic Ash Ingestion

In June 1982, the Galunggung Volcano on the island of Java erupted. A Boeing 747 encountered the volcanic debris and experienced flame-outs on three engines while the aircraft was at 33,000 feet. One engine was successfully restarted and an uneventful two-engine landing was accomplished [Lloyd 1990].

```
; Galunggung, 1982
(setf *case20*
 '(
 ; -- Solution Data
; (fault i-m-volcanic-ash-ingestion)
; (events m-group
; (1 m-causal-event
; (ante i-m-volcanic-ash-ingestion)
; (cnsq i-m-fan-blade-damage))
; (2 m-causal-event
;      (ante i-m-fan-blade-damage)
;      (cnsq i-m-fan-rotor-imbalance))
; (3 m-causal-event
;      (ante i-m-fan-rotor-imbalance)
;      (cnsq i-m-n1))
; (4 m-causal-event
;      (ante i-m-fan-rotor-imbalance)
;      (cnsq i-m-n2))
; (5 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-egt)))


 (id "Galunggung")
 (date "June 1982")
 (aircraft "Boeing 747")
 (engine "P&W JT9D-7As")
 ; -- Context Variables
 ; -- Symptoms

 (glow-in-engines i-m-visible)
 (n1 m-group
 (1 m-sensor-reading (status i-m-low))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-low))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (egt m-group
 (1 m-sensor-reading (status i-m-low))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))
 (epr m-group
 (1 m-sensor-reading (status i-m-low))
 (2 m-sensor-reading (status i-m-low))
 (3 m-sensor-reading (status i-m-low))
 (4 m-sensor-reading (status i-m-low)))

)
)
```

B.17 Massive Water Ingestion

On April 4, 1977 a Southern Airways DC-9 (Flight 242) crashed in New Hope, Georgia [NTSB-AAR-78-3]. The aircraft had flown through heavy thunderstorms and had lost both engines. The crew attempted an emergency landing on a highway and crashed. The NTSB determined that massive water ingestion into the engines accompanied by thrust level movement induced severe stalling in and major damage to the engine compressors. The NTSB determined that the aircraft might have been able to survive the weather had the flight crew not made significant movements in the thrust level.

```
; Southern Airways F-242
; NTSB-AAR-78-3
(setf *case21*
 '(
 ; -- Solution Data
; (fault i-m-water-ingestion)
; (events m-group
; (1 m-causal-event
; (ante i-m-water-ingestion)
; (cnsq i-m-fan-blade-damage))
; (2 m-causal-event
;      (ante i-m-fan-blade-damage)
;      (cnsq i-m-fan-rotor-imbalance))
; (3 m-causal-event
;      (ante i-m-fan-rotor-imbalance)
;      (cnsq i-m-n1))
; (4 m-causal-event
;      (ante i-m-fan-rotor-imbalance)
;      (cnsq i-m-n2))
; (5 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-egt))
; (6 m-causal-event
; (ante i-m-n2)
; (cnsq i-m-fuel-flow)))
; (7 m-causal-event
; (ante i-m-??)
; (cnsq i-m-epr)))

 (id "Southern Airways F-242")
 (date "April 4, 1977")
 (aircraft "DC-9")
 ; -- Context Variables
 (temp i-m-freezing)
(temp i-m-freezing)ms
; -- Symptoms
 (n1 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low)))
 (egt m-group
 (1 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (2 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low)))
 (epr m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-normal))
 (3 m-sensor-reading (status i-m-low) (trend i-m-decreases))
 (4 m-sensor-reading (status i-m-low)))
 (fuel-flow m-group
 (1 m-sensor-reading (status i-m-normal))
 (2 m-sensor-reading (status i-m-normal))
 (3 m-sensor-reading (status i-m-normal))
 (4 m-sensor-reading (status i-m-low) (trend i-m-decreases)))

)
)
```

B.18 Ice Ingestion

```
; Hypothetical scenario
;
(setf *case51*
 '(

 ; -- Fault
 (fault i-m-ice-ingestion)
 (events m-group
 (1 m-causal-event
 (ante i-m-ice-ingestion)
 (cnsq i-m-fan-blade-damage))
 (2 m-causal-event
 (ante i-m-fan-blade-damage)
 (cnsq i-m-fan-rotor-imbalance))
 (3 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-fan-vib))
 (4 m-causal-event
 (ante i-m-fan-rotor-imbalance)
 (cnsq i-m-n1))
 (5 m-causal-event
 (ante i-m-fan-vib)
 (cnsq i-m-n2)))

 (id "Hypothetical scenario 51")
 (date "Hypothetical scenario 51")
 (airline "Hypothetical scenario 51")
 (flight "Hypothetical scenario 51")
 (aircraft "Hypothetical scenario 51")

 ; -- Context Variables
 (phase-of-flight i-m-ground-start)
 (weather i-m-clear)
 (workload i-m-moderate)
 (engine-commanded-status i-m-start)
 ; -- Symptoms
 (n1 m-group
 (1 m-sensor-reading (status i-m-normal) (trend i-m-increases))
 (2 m-sensor-reading (status i-m-low) (trend i-m-increases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-increases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (n2 m-group
 (1 m-sensor-reading (status i-m-normal) (trend i-m-increases))
 (2 m-sensor-reading (status i-m-low) (trend i-m-increases))
 (3 m-sensor-reading (status i-m-low) (trend i-m-increases))
 (4 m-sensor-reading (status i-m-low) (trend i-m-stable)))
 (fan-vib m-group
 (1 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (2 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (3 m-sensor-reading (status i-m-high) (trend i-m-increases))
 (4 m-sensor-reading (status i-m-high) (trend i-m-stable)))

)
)
```

B.19 Ice Ingestion

```
; Hypothetical scenario
(setf *case52*
 '(
      (id "Hypothetical scenario 52")
      (date "Hypothetical scenario 52")
      (airline "Hypothetical scenario 52")
      (flight "Hypothetical scenario 52")
      (aircraft "Hypothetical scenario 52")

 ;                                    -- Context Variables
      (phase-of-flight i-m-ground-start)
      (weather i-m-clear)
       (workload i-m-moderate)
       (engine-commanded-status i-m-start)
                                     ; -- Symptoms

      (compressor-vib m-group
            (1 m-sensor-reading (status i-m-high))
            (2 m-sensor-reading (status i-m-high)))
      (thrust m-group
             (1 m-sensor-reading (status i-m-low))
            (2 m-sensor-reading (status i-m-low) (trend i-m-increases)))
      (fan-vib m-group
             (1 m-sensor-reading (status i-m-high))
            (2 m-sensor-reading (status i-m-high)))
      (fuel-flow m-group
            (1 m-sensor-reading (status i-m-low))
             (2 m-sensor-reading (status i-m-low)))
      (n2 m-group
            (1 m-sensor-reading (status i-m-low))
            (2 m-sensor-reading (status i-m-normal)))
      (egt m-group
            (1 m-sensor-reading (status i-m-low))
            (2 m-sensor-reading (status i-m-low)))

)
)
```

Bibliography

[1] AAIB-AAR-4/90, Air Accidents Investigation Branch. (1990). *Report on the accident to Boeing 737-400 G-OBME near Kegworth, Leicestershire on 8 January 1989.* AAIB-AAR-4/90.

[2] Abbott, K. (1990), *Robust Fault Diagnosis of Physical Systems in Operation.* PhD Thesis, Rutgers University.

[3] Alexander, P., Minden, G., Tsatsoulis, C., and Holtzman, J. (1989). Storing Design Knowledge in Cases. In *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

[4] Alterman, R. (1986). An adaptive planner. In *Proceedings of AAAI-86*, pages 65-69, American Association for Artificial Intelligence, Morgan Kaufmann Publishers, Inc., Los Altos, California.

[5] Bain, W. M. (1986). *Case-Based Reasoning: A Computer Model of Subjective Assessment.* PhD thesis, Yale University.

[6] Bain, W. M. (1986b). Assignment of Responsibility in Ethical Judgments. In *Experience, Memory, and Reasoning.* Kolodner J. L., and Riesbeck, C. K., editors. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

[7] Buchanan, B. G., and Shortliffe, E. H., editors (1984). *Rule-Based Expert Systems.* Addison-Wesley Publishing Co., Readings, MA.

[8] Chandrasekaran, B., and Mittal, S. (1982). Deep Versus Compiled Knowledge Approaches to Diagnostic Problem-Solving. In *Proceedings of AAAI-82*, pages 349-354, American

Association for Artificial Intelligence, Morgan Kaufman Publishers, Inc., Los Altos, California.

[9] Charniak, E., and McDermott, D., (1985). *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Co., Readings, MA.

[10] Charniak, E., Riesbeck, C. K., McDermott, D., and Meehan, J. R. (1987). *Artificial Intelligence Programming Techniques.* Lawrence Erlbaum Associates, Hillsdale, N.J., second edition.

[11] Davis, R. (1984).Diagnostic Reasoning Based on Structure and Behavior. *Artificial Intelligence*, vol. 24, pages 347-410.

[12] De Kleer, J. and Brown, S. J. (1985). A Qualitative Physics Based on Confluences. In *Qualitative Reasoning about Physical Systems.* Bobrow, D. G., editor. MIT Press, Cambridge, Massachusetts

[13] Feyock, S., and Li, D. (1990). Simulation-Based Reasoning about the Physical Propagation of Fault Effects. In *Proceedings of 1990 Goddard Conference on Space Applications of Artificial Intelligence.*

[14] Feyock, S. (1991). *Automatic Determination of Fault Effects on Aircraft Functionality.* Midgrant Report 1990, NASA grant NCC-1-122, February 1991.

[15] Feyock, S., and Karamouzis, S. (1991). Design of an Intelligent Information System for In-Flight Emergency Assistance. In *Proceedings of 1991 Goddard Conference on Space Appli-*

cations of Artificial Intelligence.

[16] Feyock, S., and Karamouzis, S. (1991). *LIMAP.* Technical Report, College of William & Mary, Computer Science Department, in preparation.

[17] Stefan Feyock and Stamos Karamouzis, S. (1992) A Path-oriented Matrix-Based Knowledge Representation System, in *Proceedings of the 1992 IEEE International Conference on Tools with Artificial Intelligence,* Arlington, Virginia, November 1992.

[18] Forbus, K. D. (1985). Qualitative Process Theory. In *Qualitative Reasoning about Physical Systems.* Bobrow, D. G., editor. MIT Press, Cambridge, Massachusetts

[19] Goel, A. (1989). Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem-Solving. PH.D. Thesis, Ohio State University.

[20] Goel, A., and Chandrasekaran, B., (1989). *Use of Device Models in Adaptation of Design Cases.* Technical Research Report 89-AG-DESIGNCASE, The Ohio State University, Laboratory for Artificial Intelligence Research.

[21] Goel, A., (1991). *KRITIK: An Integrated Design System; Combining Model-Based and Case-Based Reasoning* Technical Report GIT-CC-91/15, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[22] Goel, A., and Stroulia, E., (1991). *KRITIK2: Model-Based Reasoning in the Context of Experience-Based Design* Technical Report GIT-CC-91/16, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[23] Goel, A., Kolodner, J., Pearce, M., Billington, R., and Zimring, C., (1991). *ARCHIE: A Case-Based Architectural Design System.* Technical Report GIT-CC-91/18, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[24] Hammond, K. (1988). CHEF: A model of case-based planning. AAAI-86, 267-271.

[25] Hammond, K. (1989). Case-based Planning: Viewing Planning as a Memory Task. *Perspectives in Artificial Intelligence.* Academic Press, Boston, MA

[26] Hayes, P. J. (1979). The naive Physics Manifesto. In *Expert Systems in the Microelectronics Age.* Michie, D., editor. Edinburgh University Press, Edinburgh.

[27] Hammond, K. and Hurwitz, N. (1988). Extracting Diagnostic Features from Explanations. In *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

[28] Kass, A. (1986). Modifying explanations to understand stories. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society,* pages 691-696, Cognitive Science Society, Lawrence Earlbaum Associates, Hillsdale, New Jersey.

[29] Kokinov, B. (1988). Associative Memory-Based Reasoning: How to represent and retrieve Cases. In *Artificial Intelligence III: Methodology, Systems, Applications.* North-Holland, Netherlands.

[30] Kolodner, J. L. (1983). Towards an Understanding of the Role of Experience in the Evolution from Novel to Expert. In *International Journal of Man-Machine Studies,* Vol. 19.

[31] Kolodner, J., Simpson, R. and Sycara-Cyranski, K. (1985). A process model of case-based reasoning in problem solving. In *The Ninth International Conference on Artificial Intelligence.*

[32] Kolodner, J. L. and Kolodner, R. M. (1987). Using Experience in Clinical Problem Solving. *IEEE Conference on Systems, man, and Cybernetics.*

[33] Kolodner, J. L. (1988). Retrieving Events from a Case Memory: A Parallel Implementation. In *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

[34] Kolodner, J. L. and Simpson, R. L. (1988). *The MEDIATOR: A Case Study of a Case-Based Problem Solver.* Technical Report GIT-ICS-88/11, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[35] Kolodner, J. L. (1990). *An Introduction to Case-Based Reasoning* Technical Report GIT-ICS-90/19, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[36] Koton, P. (1988). Reasoning about Evidence in Causal Explanations. In *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

[37] Koton, P. (1989). *Using Experience in Learning and Problem Solving.* Ph.D. Thesis MIT/LCS/TR-441. MIT Press, Cambridge, Massachusetts

[38] Kuipers, B. (1985). Commonsense Reasoning about Causality: Deriving Behavior from Structure. In *Qualitative Reasoning about Physical Systems.* Bobrow, D. G., editor. MIT

Press, Cambridge, Massachusetts

[39] Lebowitz, M. (1980). *Generalization and Memory in an Integrated Understanding System.* Research Report 186, Yale University.

[40] Lebowitz, M. (1983). Generalization from Natural Language Text. *Cognitive Science,* Vol. 7 No. 1.

[41] Lloyd, A. T. (1990). Vulcan's Blast. *Airliner,* April-June 1990.

[42] Michie, D. (1971). Formation and Execution of Plans in Matching. In *Artificial Intelligence and Heuristic Programming.* Finoler & Meltzer, editors, American Elsevier.

[43] Minsky, K. (1975). A framework for representing knowledge. In *The Psychology of Computer Vision.* Winston, P. H., editor, McGraw-Hill, New York.

[44] NTSB-AAR-72-9,National Transportation Safety Board. (1972). *Aircraft Accident Report: United Airlines, Inc., Boeing 737-222, N9005U, Philadelphia International Airport, Philadelphia, Pennsylvania, July 19, 1970.* NTSB-AAR-72-9.

[45] NTSB-AAR-75-2,National Transportation Safety Board. (1975). *Aircraft Accident Report: National Airlines, Inc., DC-10-10, N60NA, Near Albuquerque, New Mexico, November 3, 1973.* NTSB-AAR-75-2.

[46] NTSB-AAR-76-19,National Transportation Safety Board. (1976). *Aircraft Accident Report: Overseas National Airways, Inc., Douglas DC-10-30, N1032F, John F. Kennedy Interna-*

tional Airport, Jamaica, New York, November 12, 1975. NTSB-AAR-76-19.

[47] NTSB-AAR-78-3,National Transportation Safety Board. (1978). *Aircraft Accident Report: Southern Airways Inc., DC-9-31, N1335U, New Hope, Georgia April 4, 1977.* NTSB-AAR-78-3.

[48] NTSB-AÅR-79-17,National Transportation Safety Board. (1979). *Aircraft Accident Report: American Airlines Inc., DC-10-10, N110AA, Chicago-O'Hare International Airport, Chicago, Illinois, May 25, 1979.* NTSB-AAR-79-17.

[49] NTSB-AAR-81-10,National Transportation Safety Board. (1981). *Aircraft Accident Report: Northwest Airlines 79, McDonnell Douglas DC-10-40, N143US, Leesburg, Virginia, January 31, 1981.* NTSB-AAR-81-10.

[50] NTSB-AAR-82-3,National Transportation Safety Board. (1982). *Aircraft Accident Report: Air Florida Airlines Inc., McDonnell Douglas DC-10-30CF, N101TV, Miami, Florida, September 22, 1981.* NTSB-AAR-82-3.

[51] NTSB-AAR-82-5,National Transportation Safety Board. (1982). *Aircraft Accident Report: Eastern Airlines Flight 935, Lockheed L-1011-384, N309EA, Near Colts Neck, New Jersey, September 22, 1981.* NTSB-AAR-82-5.

[52] NTSB-AAR-82-08,National Transportation Safety Board. (1982). *Aircraft Accident Report: Air Florida Airlines Inc., Boeing 737-222, N62AF, Collision with 14th Street Bridge, Near Washington National Airport, Washington, D.C. January 13, 1982.* NTSB-AAR82-08.

[53] NTSB-AAR-86-03,National Transportation Safety Board. (1986). *Aircraft Accident Report: China Airlines, Boeing 74SP, N4522V, 300 Nautical Miles Northwest of San Francisco, California, February 19, 1985.* NTSB-AAR-86-03.

[54] Rissland, E. L. and Ashley, K. D. (1988). Credit Assignment and the Problem of Competing Factors in Case-Based Reasoning. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 327-344, Morgan Kaufman Publishers, Inc., Los Altos, California.

[55] Riesbeck, C. K. (1988). An Interface for Case-Based Knowledge Acquisition. In *Proceedings of the DARPA Workshop on Case-Based Reasoning*, pages 312-326, Morgan Kaufman Publishers, Inc., Los Altos, California.

[56] Riesbeck, C. K., and Schank, R. C. (1989). *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

[57] Rosenberg, R. C., and Karnopp, D.C. (1983). *Introduction to Physical System Dynamics.* McGraw-Hill, New York.

[58] Schank, R. C., Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

[59] Schank, R. C., (1982). *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge University Press.

[60] Schank, R. C., (1986). *Explaining Patterns: Understanding Mechanically and Creatively.* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

[61] Schutte, C. P., Abbott, H. K., and Ricks R. W. *A Performance Assessment of a Real-time Diagnostic System for Aircraft Applications*. NASA Langley internal document, NASA Langley, Hampton, Virginia.

[62] Shinn, H. S., (1989). *A Unified Approach to Analogical Reasoning*. Technical Report GIT-ICS-90/11, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[63] Shontz, W. D., Records, R. M., and Antonelli, D. R. (1992). *Flight Deck Engine Advisor Final Report*, NASA Contractor Report 189562, Langley Research Center, Hampton, Virginia.

[64] Simmons, R. G. (1988). A theory of debugging. In *Proceedings of the First Case-Based Reasoning Workshop*, pages 388-401, Morgan Kaufman Publishers, Inc., Los Altos, California.

[65] Simpson, R. L. (1985). *A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation*. Technical Report GIT-ICS-85/18, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[66] Stanfill, C. (1987). Memory-Based Reasoning Applied to English Pronunciation. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington.

[67] Stanfill, C., and Waltz, D. (1986). Toward Memory-Based Reasoning. *Communications of the ACM*. Vol. 29, No. 12, pages 1213-28.

[68] Sussman, G. (1985). A Computer Model of Skill Acquisition. *Artificial Intelligence Series*, Volume 1, American Elsevier, New York.

[69] Sycara, E. P. (1987). *Resolving Adversarial Conflicts: An Approach to integrating Case-Based and Analytic Methods.* Technical Report GIT-ICS-87/26, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[70] Sycara, K. and Navichandra, D. (1989). Integrating Case-Based Reasoning and Qualitative Reasoning in Design. In AI in Design. Gero J. editor.

[71] Turner, E. H. (1989). *Integrating Intention and Convention to Organize Problem Solving Dialogues.* Technical Report GIT-ICS-90/02, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[72] Turner, R. (1988). Organizing and Using Schematic Knowledge for Medical Diagnosis. In *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

[73] Turner, R. (1989). *A Schema-Based Model of Adaptive Problem Solving.* Technical Report GIT-ICS-89/42, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, Georgia.

[74] Whitaker, L., Wiggins, S., Klein, G. (1989). Using Qualitative or Multi- Attribute Similarity to Retrieve Useful Cases from a Case Base. In *Proceedings of the DARPA Workshop on Case-Based Reasoning.*

[75] Wilensky, R. (1986). *Common LISPcraft.* W.W. Norton & Company, Inc. New York.

[76] Winston, P. H. (1984). *Artificial Intelligence* Addison-Wesley Publishing Co., Readings, MA, second edition.

[77] Winston, P. H., and Horn, B. K. (1984). *Lisp.* Addison-Wesley Publishing Co., Readings, MA, second edition.

[78] Yuasa, T., and Hagiya, M. (1986). *Introduction to Common Lisp.* Translated by Weyhrauch, R., and Kitajima, Y. Academic Press, Inc. Orlando, FL.

VITA

Stamos T. Karamouzis

Born in Kozani, Greece, March 31, 1963. Graduated from the 1st Lyceum in Lamia, Greece, June 1980, B.A., Pedagogic Academy of Thessaloniki, Greece, September 1982, B.S. Christopher Newport College, Virginia, January 1986, M.S. The College of William & Mary, Virginia, December 1988. Ph.D.Candidate in Computer Science, The College of William & Mary, Virginia. The acceptance of this dissertation, An Integration of case-Based and Model-Based Reasoning and its Application to Physical System Faults, will complete the requirements for this degree.

The author was a graduate research assistant for the College of William & Mary in Virginia from January 1989 until August 1993.