Design and Implementation of Automotive Model-Based Diagnosis Using the Echidna Constraint Reasoning System

by

Afwarman Manaf

B.Sc., Bandung Institute of Technology, 1986
(Indonesia)

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in the School
of
Computing Science

Approval

Name : Afwarman Manaf

Degree : Master of Science

Title of Thesis : Design and Implementation of Automotive Model-Based Diagnosis Using the Echidna Constraint Reasoning System

Examining Committee : Dr. Veronica Dahl, Chairperson

Dr. William S. Havens
Senior Supervisor

Dr. John D. Jones
Supervisor

Dr. Lou Hafer
Supervisor

Dr. Fred Popowich
External Examiner

Date Approved: 25 April 1991

Title of Thesis/Project/Extended Essay

Design and I plementation of Automotive Model-Based Diagnosis Using the

Echidna constraint Reasoning System.

Author:

(signature)

Afwarman Manaf

(name)

May 28, 1991

(date)

Abstract

Developing engineering applications using expert systems has been an important driving force for research in expert system technology. The automotive diagnosis problem is one of the most challenging domain applications. Existing automotive diagnosis systems which rely solely on rule-based knowledge inherit many limitations. Model-based diagnosis is an area of active research which overcomes these limitations since it uses knowledge from first principles.

This thesis provides a new model-based diagnosis system for automotive systems. Our architecture uses a deductive model as the main part and diagnostic rules to increase the speed of diagnosis processes. We use both abductive and deductive reasoning to provide a more powerful diagnostic system. Both model-based techniques and heuristic knowledge are integrated in the system. We claim that our model makes a significant step towards the use of a generic model to solve problems in many applications. The model integrates both correct and faulty behaviour using first-order logic and has a high degree of readibility and modifiability. We also use constraint satisfaction techniques that have been shown to increase efficiency in AI applications.

We have developed a working prototype of a model-based diagnostic system. The system generates the list of faulty components based on observational input supplied to it. We use the Echidna reasoning engine, a new expert system shell being developed in the SFU Expert System Laboratory.

Acknowledgements

It is my pleasure to acknowledge the many people involved through the completion of this thesis. My senior-supervisor, Dr. Bill Havens, provided many hours, insight, discussion and encouragement during our weekly meetings. His support and help were always there when I needed them. Dr. John Jones helped me to understand the other side of the issues I was tackling. He also patiently helped and corrected me in writing this thesis. Dr. Lou Hafer provided much encouragement.

I especially want to acknowledge Charlie Hunter who deserves credit. He helped me in understanding the domain of my work and was very cooperative when I was building the model designed. I would like to thank to Stefan Joseph and people in McCarney Technolgy, Inc. I also thank to many good people in Expert System Laboratory, SFU; Sue Sidebottom, Miron Cuperman and Rod Davison were very helpful. I am also grateful to the numerous other students, faculty members and staff of the School of Computing Science SFU that helped me to enjoy my time as a Master student.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Knowledge about how something is supposed to work is very useful in determining why it has stopped working correctly. Model-based reasoning is based on the simple notion above. This has been an active research area particularly for the applications of diagnosis and troubleshooting [Davis,88].

Model-based diagnosis, which uses knowledge from first principles is more powerful than conventional rule-based systems [Reiter,87], [deKleer,87]. This approach reduces problems that conventional systems have such as limited coverage in symptom-cause relationships and poor performance of the system when dealing with problems which lie on the periphery of its knowledge [Abu-Hanna,88]. Model-based architecture results in a more robust, reusable and maintainable diagnostic system [Lee,90].

Diagnosis is driven by discrepancies between observations and predicted behaviour. Hypotheses are generated based on knowledge about internal processes and components' interrelationships. This notion is very natural in diagnosis problem solving; it enables us to design a system that diagnoses elegantly.

Rule-based diagnosis techniques, as have been used in some commercial systems, have major intrinsic limitations. Abductive rules are based on a weak notion of association between a set of observations and faults. It is very unlikely to provide complete coverage of symptom-cause associations, when there is an abundance of data such as data from sensors of a vehicle's electronic control module (ECM). Rule-based expert systems typically have degraded performance when they are faced with problems on the periphery of system knowledge. This type of diagnosis system must rely on heuristics derived from relationships between symptoms and faults and requires carefully tuned heuristic rules expressing domain knowledge. Rule-based diagnosis systems do not consider the device's physical structure and rely on abductive rules alone.

Model-based systems are expected to be better than the rule-based ones because model-based systems use a model describing the structure and function of the diagnosed objects. They are based on a theory associating the device's behaviour with its design or structure. A model-based diagnostic system ensures a high degree of confidence because a model can capture the essential features and structure of domain. It reasons from first principles so that it knows the device's internal processes and uses that knowledge to determine which state of the device matches actual observations. The main disadvantage of this approach is the computational inefficiency. The model-based diagnostic system tends to be very expensive because it uses search method and generate too many alternatives. Many researchers have been working in the area of model-based diagnosis systems, some of them are [deKleer,87], [Abu-Hanna,88], [deKleer,89], [Struss,89], [Hamscher,90a]. This thesis presents research results in the design and implementation of a model-based system for automotive diagnosis.

1.1. Background

Most passenger vehicles now use a computer to control aspects of the engine. The computerized control system serves to improve engine performance in general and fuel economy in particular. It also maintains good exhaust emission environmentally. The automobile control system continuously improves. This yields a better performance and more complete functionality but at the same time it becomes increasingly difficult for mechanics to diagnose engine problems. A vehicle diagnosis system is needed to help mechanics by providing a recommended procedure to fix specific types of problems. Although engine troubleshooting is known to be a hard task, expert systems have been developed to aid in troubleshooting. Some existing expert systems are surveyed in Chapter 2.

Vehicle diagnosis is a promising application of expert systems [Klausmeier,86]. Using conventional rule-based expert systems to solve the engine diagnosis problem has proven to be a complex task [Fink,86], [Tomikashi,87]. The abundance of sensors and actuators involved in the engine control system is difficult to handle by using a rule-based approach, because it is almost impossible to cover the association between sensor's data and symptoms completely. Despite the difficulty of developing such a system, the expert system approach for vehicle diagnosis is very appealing. By taking advantage of knowledge about the engine, we can build a structural and behavioural model to develop a better diagnosis system.

1.2. Proposed Work

We adopt the notion of diagnosis as the process of finding the causes of any discrepancy between a device's correct behaviour and the observed behaviour [Genesereth,82], [Reiter,87]. The device could be a whole system, a subsystem or a physical primitive component. The model is a description of structure and behaviour. It also contains specific relationships among device components or a theory about the device that must hold in certain states. Component interrelationships could be relationships such as physical values passed between components or correlations between two adjacent components derived from the laws of physics.

Diagnosis can also be seen as a process of model refinement according to observation or evidence [deKleer,84]. Any observation may lead to a discrepancy with the model. This discrepancy forces the model to be refined in order to match reality. By model refinement, we mean that the component's state responsible for the discrepancy is changed into a new state that matches observation. This process, known as hypothesis generation, is fundamental to diagnosis.

Another major task in diagnosis is how to distinguish among the hypotheses (diagnoses). The problem here is to obtain the most valuable information from more than one possible additional measurement at the lowest cost. We would like to make additional observations to refine current diagnoses and incorporate cost functions to decide which measurements would discriminate candidates most efficiently. A cost factor is taken into consideration in selecting measurements to localize the faults.

We adopt the idea of differential diagnosis that is commonly used in medical applications. Differential diagnosis is the method of finding a correct diagnosis by ruling out all but one of the possible diagnoses. It enables a program to solve problems without requiring complete observational data [Buchanan,84]. We assume that only one fault can occur at one time. Multiple faults can be seen as a combination of single fault [Hamscher,90a]. This assumption is reasonable, although not always true in reality [deKleer,90c]. Some differential diagnosis sets are provided and the program proceeds sequentially through each possibility. This method imposes an assumption to solve the problem. The main disadvantage with this method is the system does not consider multiple faults. It only evaluates each suspected diagnosis independently. However, this method greatly increases efficiency by reducing a large set of candidates into a sequence of hypotheses involving a smaller set of candidates [Pople,82].

In this work, we propose a basic diagnostic architecture as shown in Figure 1-1. This basic schema is the refined architecture described in our preliminary work [Havens,90a].

The actual automobile engine is represented as an artifact. The system knows real engine conditions from observations supplied through a hardware interface. The observation includes vector of measurements such as temperatures, pressure at particular hoses, or voltages at certain wires. All data about the artifact are interpreted subject to the type of vehicle and year of manufacturing. McCar-EAS is the data acquisition system that has been developed to provide this real-time data by McCarney Technology, Inc.. This system is designed to analyze sensors and actuators accessed by the electronic control module. It is a system running on IBM AT's and has the ability to collect, record and display data from internal combustion engines [Joseph,89]. In our case, data measurements are recorded as data files to support model-based diagnosis.

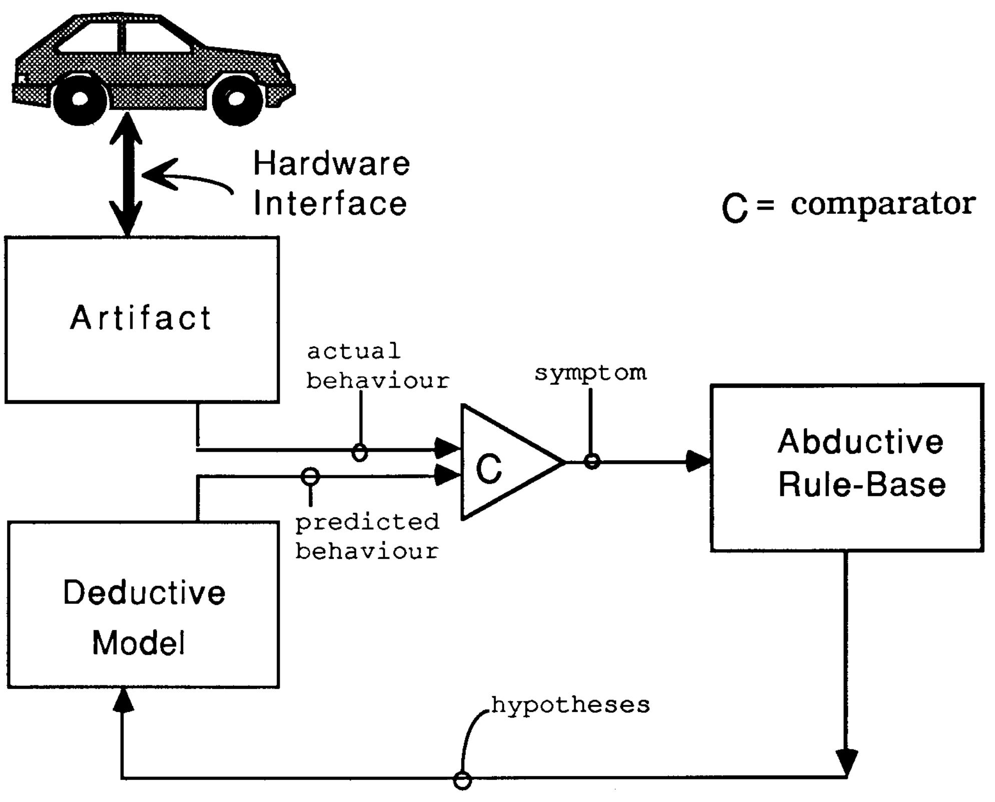

Figure 1-1
Diagnostic System Architecture

Rule-based expert systems use abductive reasoning. Given the symptoms, they guess what causes the symptoms. Instead, our diagnostic system uses both abductive and deductive reasoning. Our system relies mainly on deductive reasoning as the backbone of the diagnosis process. It hypothesizes the state of devices and components and then deduces the consequences. The deductive process yields a number of conditions or behaviour descriptions which must hold given the initial hypothesis. The hypothesis is retracted when it doesn't match the behaviour observed at the actual device.

We use composition hierarchy for knowledge representation. A complex object is composed of simple primitive components. A schema is a generative model of an object class. The model consists of characteristic parameter, components and constraints. Each parameter is a variable associated with a domain of possible values. The type of parameters could be a Herbrand, which means that its domain is the universe of all syntactic entities. Instances of an object share a common set of parameters, components and constraints. Constraints between schema instances form a constraint network of the system. Composition and other semantic relationships among components are specified by logical methods within the schema [Havens,90b].

Every component has functional states describing its behaviour for any possible condition. The model of component functional states is adequate and competent in some extent. The model is adequate in terms of the ability to characterize all possible states or behaviours of the artifact to some level of competency. By competent, meaning the model is a complete theory for characterizing the real object which contains all possible component states and their associated behavioural descriptions. We define all possible states for each component and use them as a complete set where the states of components can lie. This predefined set of states is the basis for diagnosis. The assumption of model competency has limitations and needs to be defined within the modelling abstraction. We can capture as much of the component's physical behaviour as is required to give sufficient diagnosis. One may argue that complete competency of the model can never be achieved. The ideal model of a component is a real component itself.

The behavioural description is a theoretical model which applies when a component is in a certain state. We can also view diagnosis as a reasoning process within the predefined possible states of the device. Deduction from this knowledge can be seen as a simple case of theory formation. We treat defaults as predefined possible hypotheses. The idea of theory formation from a fixed set of possible hypotheses is an essential and basic characterization of default reasoning [Poole,88].

The proposed default model without abductive rule-based reasoning involved is adequate to diagnose engine control related problems. The basic diagnostic theory illustrated in figure 1-1 is adequate to solve the problem in a correct and complete fashion. However, the deductive process is slow because it does not inherit knowledge of diagnostic procedure. Deduction uses 'search' in finding a solution. The search mechanism uses the generate-and-test method which naturally generates many solutions and explodes exponentially when faced with a combinatorial problem. Therefore the deductive part is not procedurally efficient. We use abductive rules to gain efficiency. Abduction is a method to guess diagnoses based on heuristics. This is more efficient than the generate-and-test method because it does not explore all the search space. However, there is no guarantee that the choice or assumption is correct. There is also no guarantee of completeness, the correct diagnosis may not even tried.

Rule-based knowledge is applied to sort out obvious problems based on the mechanic's experience. Abductive part of knowledge-base always tries to make suggestions before the deductive process takes place. The abductive part includes rules and an *a priori* probability method. Since it is derived from compiled knowledge, it is not exhaustive. We also apply a simple probabilistic method in this abductive part. When rules can not find a matched hypothesis, the system hypothesizes based on a set of *a priori* failure probabilities for each component. The abductive process is efficient but it has limited power in diagnosis. On the other hand, the deductive process is complete but slow because it explores all the search space.

A comparator is used in deductive processes. Its function is to detect any discrepancy between observed behaviour and the behaviour predicted from deductive processes. This comparator is implicit and is used to test the component's state and all deduced consequences of it against actual observations.

Our diagnosis system starts the diagnostic procedure by constructing a vehicle model and putting it in a normal state. The normal state is made as the first state chosen in each default model. When no information or complaint is supplied by the user, it is assumed that the vehicle is in its correct state, or 'normal'. Any complaint or data from actual observations can lead to discrepancies between current states and observations. This discrepancy causes the model to be refined by retracting inconsistent states of device components and assigning new ones that match observations. We apply rules to find the explanation based on abductive reasoning. Then, the system deduces all consequences of the hypothesis suggested by the rules and compares them to the actual observations. The reasoning process within the deductive model relies on a backtracking mechanism [Havens,90b]. Diagnosis is over when there is no difference between model and artifact.

The system is implemented in Echidna. Echidna supports logic programming language within the object-oriented paradigm. It supports model-based reasoning using schema knowledge representation. The k-ary arc consistency technique for constraint reasoning in Echidna help to solve the problem efficiently [Havens,90b], because it preprocesses the search space before elaborating any goal. Echidna is a constraint-based reasoning engine that uses a truth maintenance system for dependency-directed backtracking. In contrast to [GDE,87], which uses multiple contexts, Echidna uses a single-context truth maintenance system.

Our approach to diagnosis takes advantage of fault models in addition to correct behavioural models. The diagnostic procedure stops when no discrepancy occurs between predicted behaviour and the artifact. The system performs iterative diagnosis discrimination by suggesting additional measurements to localize the culprit. Another reason for the system to stop diagnosis is when no more testing or measurements are economically reasonable to perform for candidate discrimination.

1.3. Motivation

There are an increasing number of engineering applications which use expert systems. Along with the development of the Echidna reasoning engine, a new generation expert system shell, we want to develop an appropriate challenging application that can drive forward research in this area. The engine diagnosis problem is a very suitable and interesting application as a domain for a model-based system. This type of engineering problem is especially good, because it has a firm basis for evaluating system performance. Engineering generally has well-tested theoretical foundations, therefore the results of an engine diagnosis system are testable. Formal techniques in engineering have roots in commonsense reasoning, because there is a broad range of competence and many kinds of knowledge are involved [Forbus,88]. These facts make it more suitable for AI applications such as diagnosis systems.

This work attempts to achieve the following research goals. The first goal is to design an architecture for general diagnostic systems and implement it for the automobile engine diagnosis problem.

The second goal is to build an engine model based on the found architecture. The model should be generic and can be used for any application related to engines, especially design and diagnosis. This attempt is strongly recommended by Forbus[88]. We introduce a general knowledge representation of the domain which is independent of any problem.

The third goal is to demonstrate a working prototype of a model-based diagnostic system. We provide a diagnostic system for the problem which is recorded by the vehicle's electronic control module.

The fourth goal is to evaluate, explore and examine the performance of a model-based reasoning system in Echidna. We evaluate and explore needs, capability, performance and features of an expert system shell to support model-based diagnosis that are instrumental in pushing forward the on-going development of the Echidna constraint reasoning engine.

Chapter 2

Related Research and Expert System Shell

Model-based diagnosis is driven by the discrepancy between actual behaviour and predicted behaviour. The model is a theory of a complex object or device which incorporates structure, behaviour descriptions and specific relationships that must hold among its components. Diagnoses are generated by refining the model until it matches the device's actual condition. Diagnosis is typically an incremental process. We need some additional facts about the artifact to conclude better hypotheses (diagnoses). The diagnosis process is performed over several iterative cycles. We stop the diagnosis process after the suggested diagnoses are accepted, meaning that a match is obtained between predicted behaviour from the deductive model and actual behaviour from observations.

Knowledge about correct models is required by model-based diagnosis systems. However, knowledge about fault models is often very useful; the use of fault models can increase the efficiency of model-based diagnosis. We integrate fault and correct modes in the deductive model. We also use an abductive rule-base to capture some well-known ways of device failure. We use Echidna, a constraint-based reasoning engine which applies a k-ary arc consistency algorithm [Mackworth,85]. Echidna also provides a clean knowledge representation system within the object-oriented paradigm. We write our knowledge base in first-order Horn Clauses. This enables us to design knowledge bases which are more understandable and readable than other existing systems. The following is a survey of related research in the area.

2.1. Model-based Diagnosis

The notion of using design information in automated diagnosis started by Roth[67] who used the test-generation algorithm. Genesereth[82] uses device specification and operation knowledge from first principles. The program called *DART* works only based on information about correct information modes; it has no information about how the diagnosed object fails. *DART* uses the 'suspect computation' and 'test generation' technique. *DART*'s algorithm is different from conventional test generation algorithms because it also takes advantage of the hierarchical structure of the device to ensure the test generation algorithm remains manageable. The diagnosis reasoning is carried out in a pure deductive process.

The main task of diagnosis is detecting faults. Scarl[84] uses functional relationships to represent the diagnosed object. The functional relationship is used to check the consistency of sensor measurements, after information about current state is given. These relations are inverted to determine hypothetical values. Scarl's system propagates hypothetical values to detect and localize faults. It calculates the implications of any values supplied with respect to current state and checks them against each suspect in the network relationship. DeKleer[84] uses causal analysis to evaluate plausible faults. The causal analysis method basically uses component connectivity to explain the behaviour of composite systems. After substituting a faulty component it continues with causal analysis again. Causal analysis can run through "down stream" and "up stream" along the physical structure. DeKleer[87] considers input-ouput relationships between upstream-downstream components of the physical lay out. This can be done easily due to the nature of multiple context paradigm incorporated in the underlying reasoning engine. This system uses an assumption-based truth maintenance system (ATMS) to keeping track of the current solution and its related assumption. It incorporates probability and information theory into the ATMS' mechanism. *GDE+* [Struss,89] exploits contradictions between assumed correct behaviour and observations; therefore it needs to know the 'value' of a component's output in particular states.

Model-based systems embody behavioural models of a device. Behavioural descriptions are represented as input-output relationships in [Genesereth,82] and are typically characterized in terms of the hierarchical nature of devices because device structure is specified by describing its parts. *Equal* [deKleer,84] incorporates commonsense knowledge together with quantitative knowledge to predict the behaviour of a composite system. It relies solely on a qualitative model and doesn't contain enough quantitative knowledge from the design point of view. This leads to a less powerful system to find inconsistences between the predicted behaviour and observed behaviour. The predictive procedure may fail to detect conflicts. Dague[87] models the structure and behaviour of the diagnosed object in order to bridge the gap between presumed correct behaviour and actual observations. Models may fail because they are oversimplified; for example, a healthy component may sometimes behave beyond its normal model. Dague's system considers the possibility of multiple behaviour patterns. Several models are required to describe a component's behaviour. His system, called *Dedale*, uses the knowledge of a human expert to model behaviour. Dague's system uses models to reason and has a set of assumptions related to the particular symptoms given. Instead of using several models, we use a single model with modes for each component. One mode of behaviour could be manifested in more than one way. For example, a primitive component such as a fuse may have a good and bad mode and it could be in bad mode in more than one way such as shorted or open.

Genesereth[82] uses a deductive procedure within each level of a compositional hierarchy in order to perform model refinement. The symptoms are expressed as violations of expected behaviour. When a symptom is found, all parts under that level are suspected. The next step is to test each suspect by a kind of test which expects certain output given any particular input. If the output doesn't agree with expectation then the underlying part must be broken and needs to be investigated further. This process goes on level by level. His system, called *DART*, uses composition structure to describe a device and how its parts interconnect. The lowest-level parts in a compositional hierarchy are the primitive components. *GDE* of [deKleer,87] is a model-based diagnosis system that can handle multiple faults. Since it doesn't use fault models, it has a drawback, in that it provides implausible diagnoses in the real world even though they may be possible logically.

In [Genesereth,82] a high level of abstraction is used to determine the major subcomponents in which the fault lies. Scarl[84] uses frames to describe each potentially faulty component and can only deal with a single fault at one diagnostic cycle. *GDE* [deKleer,87] attempts to detect fault in a different way. Suspects are represented and manipulated in terms of minimal sets of candidates. *GDE* uses an incremental diagnosis procedure by exploiting the iterative nature of diagnosis. It is a general system and can be used for domain-independent diagnosis because of the separation between diagnosis and behaviour prediction. *GDE* combines model-based prediction and sequential measurements to localize the faults. It uses conditional probabilities based on structure. *GDE* applies a one-step lookahead technique to ensure the best next measurements. GDE uses minimum entropy calculation to select the next measurements taken. The method is to find the measurements which lead to minimal additional measurements required to localize faults. Minimum entropy is a calculation derived from probability and information theory. This calculation is incorporated in the reasoning engine. The best measurement to take is the one which leads to minimum expected entropy of the resultant candidate probabilities. *GDE* needs some known values to deduce the values of system parameters. The diagnosis stops when the probability of a faulty component is high enough or when the next measurement is too costly to perform.

Struss[89] proposed an approach that has advantages over the original *GDE*. It has the ability to prove the correctness of components and to rule out implausible diagnostic hypotheses. His system, called *GDE+*, also confirms whether malfunctioning components are consistent with observations. He argues that a component will fail in a deterministic manner instead of a completely unconstrained manner, therefore it is possible to have known fault models. *GDE+* uses extended ATMS which is capable of handling 'negation' and 'disjunction'. It incorporates fault models by extending the capability of the *GDE* reasoning engine (ATMS) with these new features. The diagnostic process is done by gathering information in a cycle that decides which correctness assumptions should be retracted. *GDE+* explains and confirms its diagnosis by analyzing whether the malfunction of components is consistent with observed behaviour. Specific knowledge about what happens when a fault occurs is required. It uses resolution rules and controls reasoning to introduce appropriate fault models. Fault models are used only when necessary.

SHERLOCK [deKleer,89] uses a theory of faulty components. It identifies failures without necessarily knowing how components fail. It uses the model of behavioural modes and probability theory about the likelihood of each mode of behaviour. In *Sherlock*, diagnostic discrimination is derived from knowledge about the likely ways a component may fail. It determines whether these failure modes are consistent with observations. There is a high chance that an unlikely diagnosis will be treated as seriously as a likely one because it does not use the knowledge of diagnosis procedure derived from the human experts. *Sherlock* uses modes to discriminate diagnoses. If one mode is supported favourably by evidence, it lowers the probability of others. A component always belongs to a certain mode. The manipulation of behavioural modes and modification to focus ATMS are the main difference between *Sherlock* and the previous system described, *GDE*. It represents candidates as a list of components with assigned modes. Components fail independently and there is prior probability of finding a component in a particular mode. In *Sherlock*, the sum of components' probability is constant, so if a candidate is eliminated the other's probability will increase. The diagnostic task is to identify behavioural modes of all components.

The limitation of the model vocabulary causes a problem in localizing every fault. [Genesereth,82] suffers from this problem. *GDE* [deKleer,87] only uses correct models and doesn't require an explicit fault model. Because *GDE* lacks important diagnostic reasoning knowledge about incorrect behaviour it also suffers from the phenomenon where a predicted faulty component may not explain symptoms. On the other hand [Struss,89] depends upon the completeness of fault models, otherwise it can give wrong diagnoses. It assumes a typical known fault model occurs first and manages to get rid of implausible diagnoses. DeKleer[89] uses correct, fault and unknown modes. Each of them may have more than one model and more than one observation can result in different good modes. It focuses on more probable diagnoses to reduce alternative diagnoses. *Sherlock* requires a modification of ATMS that can focus on more probable diagnoses. This work shows again that exploiting fault models is a very active research area in diagnosis problem. It confirms that fault models are useful to pinpoint a faulty component quickly and to determine a specific repair to the faulty components.

Combinatorial explosion is a problem which occurs in diagnosis because the program uses search mechanism and generates many alternative diagnoses. Genesereth[82] incorporates constraint propagation techniques in reasoning to alleviate the combinatorial explosion problem. The design language and diagnostic procedure used postpones the instantiation of variables as long as possible. The restricted design description used in *DART* suppresses enough detail to reduce the cost compared to the use of the full-blown description. The generate-and-test process also benefits from structural abstraction. *DART* relies heavily on the existence of a design description. Computational efficiency still remains a problem in this system, because there is a trade off arising from the generality of the system. *DART* needs to enhance its knowledge involved in stages of diagnosis. *DART* has not included the cost of testing and diagnostic value of tests in discriminating and testing hypotheses. *GDE+* [Struss,89] uses control strategies to overcome the combinatorial problem. The use of the one fault assumption also reduces this inherent problem of diagnosis.

Hamscher[90a] incorporates hierarchic diagnosis and fault models into *GDE*. He dealt with the combinatorial problem by introducing an explicit context-switching mechanism into the ATMS. His system, called *XDE*, reduces the cost of reasoning by representing structure at multiple levels of abstraction and eliminating portions of the device from consideration by localizing the problem in the simplest model first. It only proceeds into detail when needed. To avoid an implausible diagnosis, *XDE* focuses on components that are statistically plausible rather than those which are logically possible. This approach is used in addition to fault models.

XDE adds two more intermediate procedures before suggesting next probes. The first is a diagnostic decomposition procedure based on hierarchical structure. It has a 'physical hierarchy' containing all assumptions about which components are working. The decomposition process descends through this hierarchical structure, replacing an assumption of an upper-level component by the changes in the assumptions of its subcomponents. The second is a refinement procedure which uses fault models to eliminate unnecessary diagnoses. *XDE* uses some thresholds to decide whether diagnoses can proceed into refinement, decomposition, and finally probe selection step in discriminating candidates.

Similar to *GDE*, *XDE* works from the knowledge of expected output of a particular component. It only has one possible model for modes, in contrast to *GDE*. Refinement based on fault model exacerbates the combinatorial problem. *XDE* system research implies that further research is needed to find a more flexible control structure. It also suggests the need for research to confirm the efficiency of explicit context switching as compared with *GDE*.

XDE chooses the next probe by using fault models and hierarchical organization. It uses the fault models to rule out implausible suspects and reasons downward through the hierarchical organization. Assumptions about fault components only appear in the physical hierarchy and behavioural models appear in the functional hierarchy. Coordination between these two hierarchies causes a potential technical problem in addition to combinatorial complexity. *XDE* invokes the behaviour model when it contains the immediate physical subcomponents.

Our system uses a deductive model as the main system knowledge base. The deductive model gives descriptions of correct and fault models. This is considered as deep knowledge in our system. We apply rule-based knowledge before using deep knowledge. The rule-base acts as the shallow knowledge and is not necessarily exhaustive. The [Abu-Hanna,88] system has the capability of learning from deep knowledge and "compiling" the knowledge at a shallow level of abstraction. This helps in subsequent cases. The new shallow expertise regarding links between faults and symptoms is constructed after expertise is used at the deep knowledge level. It uses a model of pathological failure at lower level components.

Our diagnosis system captures engine conditions at certain periods of time. Engine conditions are assumed invariant at diagnosis time. This assumption of non-intermittent faults is reasonable, although it could be false in general. *DE* [Abu-Hanna,88] assumes there are no intermittent faults when tests are applied to check the consistency of suspected components. Hamscher[90a] uses an independent failure assumption, this is a very strong simplifying assumption, although still not as strong as the assumption of independent effect that *Mycin* [Buchanan,84] makes. The probability of a component working is the product of the possibilities that each of its subcomponents works independently. The probability of diagnoses in the *XDE* program is merely a conjunction of the assumptions. *XDE* needs two hierarchical models and it only uses fault model for heuristic method which is not necessarily exhaustive.

One advantage of model-based diagnosis is the separation between the diagnostic procedure and the design knowledge about the device. In [Genesereth,82] all device dependent information, which expresses a theory about the object, is separated from knowledge about the diagnosis process and is captured within its design description. DeKleer[87] separates the diagnosis process and the sequencing procedure to localize faults. Abu-Hanna[88] also adopts the same notion of separation between diagnosis methodology and knowledge about device.

Our system uses physical connectivity to represent the low-level organization of the device. The compositional hierarchy is used as a meta-level view of the physical structure and represents a "part of" hierarchy in physical structure. In [Hamscher,90a], the physical composition hierarchy expresses the physical structure as "part-of" relationships. It knows which component is working; a component is working if all of its subcomponents are working. The functional composition hierarchy contains behavioural models. There is a technical problem in matching reasoning in the physical and functional hierarchies in which the behavioural model lies. This is because the diagnosis proceeds through the physical organization. The physical organization does not inherit behavioural descriptions. A component's behavioural descriptions are invoked when it is suspected as faulty.

[Genesereth,84] uses a knowledge base represented as a model written in a series of propositions in a variant of prefix predicate calculus. It prefers executing tests to generating a diagnosis tree, because of the high computational cost for constructing diagnostic trees. *DE* [Abu-Hanna,88] uses *Prolog* as its reasoning engine and uses qualitative together with quantitative rules. If there are no qualitative rules which apply directly to the current suspects, it uses quantitative models of the subcomponent to further investigation. It records a history of property values of each component while a hypothesis is applied. This information is used for the next improvement of higher level behavioural rules. *DE* takes advantage of compiled knowledge before starting diagnosis sequences. The success of *DE* suggests further research on the use of the object oriented approach in diagnostic systems.

2.2. Automobile Diagnosis

The problem of automotive diagnosis has been considered particularly challenging. Although it is a difficult task, automotive diagnosis is a promising and feasible application of expert systems [Tomikashi,87]. Our goal is to provide a model-based system for automobile diagnosis. Specifically, we only deal with problems which are recorded on the electronic control module.

Klausmeier[86] explores the concept of using an expert system as an external diagnostic aid in repairs. His system attempts to diagnose engine problems using a rule-based approach. Knowledge is extracted from a drivability diagnosis manual[Chevrolet,89]. In this example, the system assumes that there is no problem in the electronic control module (ECM) or in the electronic circuit. It also requires pre-examination of mechanical problems before diagnosis proceeds. Although expert system technology seems to help mechanics in garages [Joseph,89], this work suggests that the complexity of the problem and the difficulty of establishing knowledge has delayed commercial uses of vehicle engine-related-problem diagnosis systems. Another work dealing with electronic-controlled engine diagnosis is [Tomikashi,87]. This work came up with the same conclusion; it is possible to arrange and store the knowledge of mechanics to develop such a diagnostic system.

All the automobile engine diagnosis systems above are rule-based systems. Rule-based systems have only been successful because of carefully arranged knowledge concerning the domain. They have a poor capability for explanation and their performance degrades significantly when faced with problems at the boundary of their knowledge. They contain only abductive knowledge that does not use any deep theory about the diagnosed devices. Therefore, they are best for domains lacking a theoretical basis *e.g.* medicine, business, financial domains [Buchanan,84].

Realizing the serious limitations of the rule-based approach, Fink[86] developed the integrated diagnostic method (IDM) to integrate shallow knowledge, derived from empirical experience, and deep knowledge based on the functional knowledge of physical devices. This work applies the model-based approach to diagnosing mechanical and electrical devices in general, although it was intended particularly as an electronic control engine problem troubleshooter. IDM uses two separate knowledge bases for shallow knowledge and deep knowledge. Each has its own inference engine. Coordination and control between these two separate modules is performed by a controller module. This system includes qualitative modelling of faulty and correct behaviour.

Lee[90] attempted to provide a model-based diagnostic system for engines. This system, called *Repair*, is being developed after studying various domain applications developed for repairing systems. *Repair* uses a model describing physical structure and device behaviour. It models the components of a device and the ways in which these components are linked together.

2.3. Expert System Shell

Echidna is the underlying reasoning engine we use for our diagnosis system. It allows us to represent a knowledge base in object-oriented form. It also provides a constraint reasoning engine we need to solve diagnosis problems. Echidna is a new CLP Language with objects and a dependency backtracking mechanism [Havens,91].

There are many expert system technologies available to support the automatic deduction needed in diagnostic systems. What we need is a symbolic logic system that captures the notions of logical consequence with formalism and can be done mechanically. *Prolog*, a logic programming language, is a good candidate to provide a basis for expert system technology [Merritt,89]. This is a declarative language and structured in terms of relations. This notion yields many powerful ramifications for logic programs, one of which is reasoning with non-deterministic behaviour [Pereira,87]. Although it is a declarative language, it has a procedural interpretation.

Since it is a high-level language and because of the clean formalism it has, *Prolog* is a good programming language for applications in AI, databases and engineering [Kowalski,79]. *Prolog* also seems to have a close relationship with constraint satisfaction problem solving [Nadel,90]. However, *Prolog* is known to suffer from thrashing behaviour which is a consequence of backtrack search.

There have been some attempts to overcome this problem by introducing constraints into *Prolog* such as CLP(R), *Prolog II* and *Prolog III* [Colmerauer,90]. CLP has successfully added constraints in logic programming while maintaining basic theorem of first-order predicate calculus [Cohen,90]. There are many good reasons to choose CLP languages for building diagnostic systems. The non-deterministic feature of CLP and the use of constraints makes CLP very efficient for implementing a diagnostic system [Colmerauer,90].

Data structure and knowledge representation schemes are the major considerations in diagnostic system design. The object-oriented approach allows powerful knowledge abstraction [Meyer,88]. The "objects" in our model reflect real world objects. We need to develop operation that are applicable to certain objects and specify the effect of each operation. The object-oriented approach seems to be beneficial for diagnostic system construction design.

2.3.1. Echidna

Echidna is a new generation of expert system shell that provides object-oriented representation and embeds constraints in logic programming [Havens,90b]. It uses active constraint reasoning techniques to detect failure in the search more efficiently. When failure is detected, it uses a dependency-directed backtracking algorithm to avoid unnecessary computation. Echidna is a CLP language with an object-oriented schema representation. Echidna is a good choice to implement our diagnostic system, because it fits very well with the natural ontology of constraint-based hypothetical reasoning that we use in our system [Havens,90b], [Havens,91]. It is the only available language that provides constraint reasoning techniques with a good schema representation in our knowledge.

Echidna is a theorem prover which deals with Horn Clauses. A clause is an expression of the form H:- B1, B2, ..., Bn. where H and B1, B2, ..., Bn. are terms. H is the head and B1, B2, ..., Bn. are the subgoals. For example :

"state(good):- VoltIn =:= VoltOut, CurrentIn =:= CurrentOut."

Unification is a substitution which makes the two terms identical such as state(State) and state(good). These terms unify because they have common unifier { State = good }. Echidna applies an interpreter to answer queries issued. The interpreter generates 'yes' when the query succeeds and 'no' when the query is not deducible from the program.

Knowledge of particular domain is represented in the form of schemata. Schemata is an object-oriented programming language paradigm adapted to CLP. The following is an example of a simple Echidna knowledge base.

```
voltRange = {0..24}.
currentRange = {-20..20}.

schema wire:component
{
% type declarations
        voltageRange Volt1. currentRange CurrentIn1.
        voltageRange Volt2. currentRange CurrentOut2.
        volt1(Volt1). currentIn1(CurrentIn1).
        volt2(Volt2). currentOut2(CurrentOut2).

% accessors for persistant variables
        terminal1(Volt1,CurrentIn1).
        terminal2(Volt2,CurrentOut2).

% modes of operation
        order mode.
        mode:- % good state
                Volt1 =:= Volt2,
                CurrentIn1 =:= CurrentOut2,
                State =:= 0,
                Condition =:= good.

        mode:- % shorted state
                Volt1 =:= Volt2,
                CurrentIn1 =\= CurrentOut2,
                State =:= 1,
                Condition =:= bad.

        mode:- % open state
                CurrentIn1 =:= 0,
                CurrentOut2 =:= 0,
                State =:= 2,
                Condition =:= bad.
}
```

Figure 2-1
The Echidna Schema

A variable in Echidna is a sequence of characters which begin with an upper case letter or underscore (_). A Variable is bound to a domain over which the variable ranges when it is referenced at the first time. A variable can be over an Herbrand and also it can ranges over discrete and real intervals. For example :

```
{good,shorted,open}      State.
voltRange                VoltIn.
```

Objects in knowledge base are organized into classes. Each class is defined by a schema. A schema is a model for a real object and can be instantiated when needed. Each schema may contain definitions or methods. The more details about Echidna is explained throughout this thesis.

Chapter 3
Diagnosis System Design

3.1. System Overview

In this chapter, we present a model-based expert system for diagnosing computer-controlled engine malfunctions. The system uses a knowledge base of engine models. Our model includes the description of correct and faulty engine behaviour. We also incorporate rule-based knowledge to improve system performance. This abductive rule-base uses heuristics and *a priori* likelihood of failures of each component. In fact, our system's knowledge-base has both deep and shallow knowledge. The deep knowledge is the deductive model which includes correct and faulty behaviour. The shallow knowledge is an abductive rule-base based on heuristics.

3.1.1. Model-based Diagnosis

Model-based diagnosis is carried out using the interaction between predictions and observations. Observations come from the measurements taken or complaints of the car owner. Predictions are the behavioural descriptions derived from the assumption within the deductive model in the form which can be compared to the observations. Discrepancies between actual and predicted behaviour drive the diagnosis process which traces the possible causes of the problem described within the deductive model. For example, consider a wire. At one time the wire may be in 'good' state. (We use the terms 'state' and 'mode' interchangeably with the same meaning). The discrepancy between predictions and observations from this assumption may cause the state of the wire to be changed in the model. The change of the model is performed automatically using backtrack every time a discrepancy occurs in the deductive process.

The model-based approach is powerful, because it enables the diagnostic system to diagnose single or multiple faults for various kinds of engine [Hamscher,90b]. We can use the same architecture to diagnose any device by substituting the models with the required ones. Unfortunately the cost of reasoning is a major problem in the model-based diagnosis. It can generate too many alternatives that are logically possible. In the worst case it has complexity of $O(k^n)$ [Aho,74], where k represents the number of modes of each component and n is the number of components comprising the whole device.

We deal with the combinatorial explosion problem by using the abductive rules to eliminate the less likely alternatives. Instead of letting a backtracking mechanism explore the search space, we apply heuristics for some obvious symptoms. We use rules of thumb to guess the culprits. This strategy avoids the exhaustive backtrack search within the deductive model. If there is no rule which can give a correct 'guess', the system uses the single-fault assumption and makes a hypothesis based on the ordered probability of failures amongst components. We should make it clear that this single-fault assumption is needed because our system does not consider the effects of multiple-faults. However, the diagnostic system can still detect more than one faults whenever it is consistent to the predicted behaviour deduced in the deductive model.

We construct the engine model by defining schemata which contain methods in the form of logical horn clauses. The left-hand part is a non-negated literal that becomes the head of the clause. The right-hand part is conjoined to form the body of the clause. The possible modes of a component in the deductive model are represented as the heads of the clause. The body represents subgoals that must hold for the underlying mode. Figure 3-1 shows the behavioural description of the wire as a primitive component. There are three possible behavioural modes of wire. It could be good, shorted or open. The antecedent of the rules, which are written in Prolog-like syntax, describes the behaviour in each possible mode.

```
good:-
        voltIn =:= voltOut,
        currentIn =:= currentOut.

shorted:-
        currentIn =\= currentOut.

open:-
        currentIn =:= 0,
        currentOut =:= 0.
```

Figure 3-1
Wire's Behavioural Description

3.1.2. Generating and Discriminating Diagnoses

Diagnosis generation is the process of finding the potential faulty components given a vector of measurements or drivability complaints. A diagnosis is the collective state of every component in the engine model after the process of model refinement is completed. The following steps apply in directing the changes of component states throughout a model refinement process: The first is the heuristics which relate observations to some known possible failure of the components. By using rules, the system can direct a component to be in a good or faulty state. However, these rules are subject to error. By error, we mean the chosen rule may not explain the symptoms. The rules are not complete, because it is impossible to exhaust real-world phenomena in rules. However, we expect the chosen assumption is almost always true. The second is the single-fault-based probabilistic method which assigns certain components to be in faulty modes based on *a priori* likelihood of failures. The third step, which acts as the confirming step, is the deductive process applied to competent models. The deductive model guarantees that at least one possible mode matches the observations because the model is adequate to cover all possible conditions of the artifact.

Testing and measurement are very valuable in diagnosing engine malfunctions. They give information that enables the system to localize the faults. Unfortunately, measurements or testing are usually expensive and time-consuming. We would rather exploit computational power than make unnecessary expensive measurements. The system is pushed to pick up more measurements if the current observations do not lead to any diagnoses. Our system in this stage considers only the most likely diagnoses at one time.

We take cost factors into consideration while gathering information. The next measurement is the one that gives the most valuable information with minimal cost to performing it. This is derived from the experience of mechanics and incorporated within the shallow knowledge. The system benefits from accumulated measurements taken to discriminate diagnoses, therefore we assume that all engine states are stable throughout the diagnosis process. We always choose the vectors of measurements that haven't been given so far. For the time being, we are using data extracted from the real observation to simulate the diagnosis process.

3.2. Diagnostic System Components

3.2.1 Artifact

We gather engine data from measurements by using a hardware interface provided by McCarney Technology, Inc.. In fact McCar-EAS supplies these measurements in a data file form [Joseph,89]. We are provided with such data as temperature, voltage, current, pressure, and volume of flowing air that can be measured at several check points on the real engine. Figure 3-2 shows some real data examples and their ranges that can be measured and provided through the hardware interface. Table 3-1 shows an example of measurements taken on the 12-volt power supply system. We have eight testpoints within the 12-volt power supply system that are possible to observe for diagnosis purpose.

Parameters :	Range :	
Auto key.	0/1	on/off.
Battery voltage	0.0 - 20.0	
Bypass line voltage	0.0 - 100.0	
CCP-duty	0.0 - 100.0	
Engine cranking RPM.	0.0 -1400.	
Fuel pump position switching voltage	0.0 - 20.0	
Fuel pump primary switching voltage	6.0 - 20.0	
Manifold absolute pressure	0.0 - 104.0	
Transmission temperature	-40.0 - 180.0	

Figure 3-2
Ranges of The Real Data

12-Volt Power Supply System

#	Signal	V1	V2	V3	M1	M2	M3	M4
				(volts)				
1	off	11.9	0.0					
2	off	0.0	0.0					
3	on	0.0	0.0	11.9				
4	on	0.0	0.0	11.9				11.9
5	on	0.0	0.0	11.9			0.0	11.9

Table 3-1
Some Measurements

3.2.2. Deductive Model

The deductive model is the main part of our model-based diagnosis system. It should cover every possible state of the engine and its behavioural descriptions. The state of the engine is represented by the corporate state of the engine components. We show the model of a wire as an engine component with three possible states. The example of wire's component behavioural model is shown in figure 3-3. This deductive model embodies the knowledge about correct and fault modes of the wire.

```
voltRange = {0..20}.
currentRange = {-20..20}.
schema wire:component
{
% type declarations
        voltageRange Volt1. currentRange CurrentIn1.
        voltageRange Volt2. currentRange CurrentOut2.
        volt1(Volt1). currentIn1(CurrentIn1).
        volt2(Volt2). currentOut2(CurrentOut2).

% accessors for persistant variables
        terminal1(Volt1,CurrentIn1).     terminal2(Volt2,CurrentOut2).

% modes of operation
        order mode.
        mode:- % good state
                Volt1 =:= Volt2,
                CurrentIn1 =:= CurrentOut2,
                State =:= 0,
                Condition =:= good.

        mode:- % shorted state
                Volt1 =:= Volt2,
                CurrentIn1 =\= CurrentOut2,
                State =:= 1,
                Condition =:= bad.

        mode:- % open state
                CurrentIn1 =:= 0,
                CurrentOut2 =:= 0,
                State =:= 2,
                Condition =:= bad.
}
```

Figure 3-3

The Deductive Model of A Wire

The domain values for the voltage are real numbers ranging from 0 to 20 volts. The current varies between -20 and 20 amperes. We use a symbolic discrete domain for the state parameters of the wire. We define three possible modes in the default model of the wire. The clause is formed from "disjunctive or" literals so that the default state of the wire is to be in the good state.

The adequate model of the wire is the one that describes the good state of the wire. Therefore the otherwise behaviour is considered as a bad state. Since the knowledge of fault modes are also available in some extent, we define the possible modes of a wire to be {good, shorted,open} instead of just good or bad state. These three possible modes of the wire's deductive model will be checked non-deterministically when the deductive process takes place. We can arrange them in a particular order so that the system backtracks according to the sequence we want. This is useful to control backtracking according to knowledge of diagnosis procedure derived from experience.

3.2.3. Abductive Rule-base

The abductive part of the diagnosis system consists of heuristic rules and a probability method based on *a priori* likelihood. We use a set of rules derived from the mechanics' experience. These rules are the first step applied to detecting faults. However, abductive rules may fail to explain the symptoms. Examples of abductive rules are shown in Figure 3-4, M1 is a possible testpoint that lies close to the battery. This an obvious problem; the mechanic knows that if the voltage measured at that point is more than 10 volts, it is very likely that the battery is in a good state. If the actual data does not match with the first clause, the next clause will be choosen and so on.

We use probabilities in addition to rules. The probability method uses a simple list of components ordered by their likelihood of failure. It imposes a

```
rule:-
        Signal =:= 'on',
        M1 < 16,
        M1 > 10,
        Battery:state(good).
rule:-
        Signal =:= 'on',
        M4 < 10,
        Junction2:state(bad).
```

Figure 3-4
Heuristic Rules

specific state for components. For example, the wire could be in a bad condition either by being in the shorted state or by being open. This probability method improves the system's ability to detect failure. Figure 3-5 shows a list of ordered *a priori* probabilities for faulty components within the 12-volt power system.

```
Wire2:shorted.
Battery:undercharged.
Junction2:open.
Ig-Switch:bad.
ECM-Fuse:bad.
Engine-Fuse:bad.
```

Figure 3-5
A priori Likelihood of Failures

3.2.4. Comparator

A Comparator is applied implicitly within the system. The comparison takes place at the level of the deductive process to detect whether the predicted behaviour matches the observation. We can explain this by looking at figure 1-1. The discrepancy between the predicted and actual behaviour causes a symptom, the symptom is a binary state signal indicating the discrepancy occurence. The symptom generated by the comparator leads the abductive rule-base to hypothesize about the engine state based on heuristics. The "engine state" here means the states of the engine's components. This assumption is applied into the deductive model in order to generate the deduced behaviour. The system deduces the consequences of this assumption and compares them against observations. The comparison is implemented through unification of parameter values and applying constraint satisfaction algorithms.

3.3. Diagnosis Scenario

The automotive diagnostic system starts by building a complete vehicle engine structure. The vehicle engine structure is an instantiation of a complex object from a vehicle schema which is comprised of several subsystem schema classes. Every complex object consists of more than one object and so on. The engine structure is formed from primitive component's object at the lowest level. Primitive objects are connected by unification through their input-output parameters.

The engine structure is a complete network of primitive components that are already connected by input-output parameters. At the beginning, there is no particular state applied to the engine, no assumption is selected and no constraint is activated. The diagnosis process commences when a measurement or a complaint is supplied as the input data. Given this observational data, the system immediately applies data to the engine structure (observational data are persistent unless the user intentionally retracts them). Any specific complaint of the car owner is also recorded and will be used by the system at the abductive level.

Given the observational data, the system puts the engine model in a good state and then compares the predicted behaviour with the given data. The diagnosis process continues while a symptom occurs, otherwise the diagnosis is over. Based on the available information from measurement and complaint, the system activates applicable abductive rules and refines the deductive model heuristically. The system applies dependency-backtracking mechanism into the deductive model. Because the deductive model is adequate, the system should find diagnoses eventually.

Abductive rules impose particular states on one or more components based on observations and heuristics. The components which have been forced into a certain state yield the deduced consequences. This deduced behaviour may cause inconsistency with actual observations. If the discrepancy happens the system would retract the current states of engine components in favour of new states which match actual observations. If a rule succeeds, the process is over. Otherwise it chooses rules until one of them succeeds or the rules are exhausted. A probabilistic method which is based on single-fault assumption will be applied if the rule base can not capture the diagnosis. The probabilistic method is basically a guess directing which component is more likely to fail. The dependency backtracking mechanism triggers system failure if constraint satisfaction method fails to achieve the goal. Failures cause the system to backtrack until one of the system choices is consistent with observations.

3.4. Efficiency

Together with the deductive model which incorporates fault modes in addition to correct modes, the abductive rule-base aims to make diagnosis more efficient. The set of abductive rules is instrumental to gaining efficiency based on heuristics which is significantly faster than deduction. The use of the single-fault assumption helps to eliminate some unnecessary alternatives in hypothesis generation. The use of the abductive part in our diagnostic system improves computational efficiency which is one of the main obstacles in model-based diagnosis systems.

In the deductive part of the system the use of constraint logic programming (CLP) reduces the search space in the reasoning process. We use CLP because it has features that enable us to express constraints and manipulate them efficiently. CLP is efficient because this programming paradigm exploits constraint solving techniques for various kinds of specific domains [Jaffar,87]. The idea is to restrict domains of variables to only values which participate in the global solution of the diagnosis problem. The system's reasoning engine applies an appropriate algorithm for different computational domains throughout the diagnosis process.

We use Echidna, a CLP language, in implementing the diagnostic system. Echidna incorporates the k-ary arc consistency algorithm AC-3 of [Mackworth,77] for constraint propagation over discrete domains. For example, the condition of a wire is represented by the discrete domain {good, shorted, open}. Once the system knows that a particular wire is in a state good, it immediately eliminates the other two possible values from state's domain. A more efficient version of AC-3 is applied to handle the integer domain. Echidna introduces the ability to use constraint processing on variables which range over real intervals. Variable domains are represented explicitly as a hierarchical structured set of intervals. Equalities and inequalities are propagated through a specialized hierarchical arc consistency algorithm [Sidebottom,91a].

Another significant feature of the Echidna reasoning engine used in our system is dependency-directed backtracking [Havens,90b]. This is an intelligent backtracking technique, developed to cope with the well-known thrashing problem [Mackworth,77]. It backtracks to the cause of failure instead of to the most recent choice. It also avoids rediscovery of contradictions in backtracking by recording no-good assumptions.

3.5. Combinatorial Problems

The use of behavioural models which include correct and fault modes exacerbates the combinatorial problem [deKleer,89]. It produces $O(k^n)$ possible diagnoses, where k is the average number of components' modes and n is the number of components which comprise the whole diagnosed device. The single-fault assumption reduces the number of possible diagnoses significantly. To illustrate this, assume a system comprised of two components A and B. Each of them has four behavioural modes which include correct and fault modes. This system has only one 'good' state and more than one non-good state. Sa_i represents the possibility component

Sa1	Sb1	Sa3	Sb1
Sa1	Sb2	Sa3	Sb2
Sa1	Sb3	Sa3	Sb3
Sa1	Sb4	Sa3	Sb4
Sa2	Sb1	Sa4	Sb1
Sa2	Sb2	Sa4	Sb2
Sa2	Sb3	Sa4	Sb3
Sa2	Sb4	Sa4	Sb4

multiple-faults : $O(k^n)$

Sa1	Sb1
Sa1	Sb2
Sa1	Sb3
Sa1	Sb4
Sa2	Sb1
Sa3	Sb1
Sa4	Sb1

single-fault : $O((k-1)n)$

Figure 3-6
The Possible Diagnoses

"A" being in state i. Figure 3-6 shows all possible diagnoses based on the multiple-faults assumption compared to the single-fault assumption. The diagnosis search remains complete, even with the single-fault assumption, because this assumption applies only to the abductive step; the deductive model can still diagnose multiple faults. We argue that this assumption is more reasonable for a diagnosis system which use abductive rules to alleviate the combinatorial problem. We only use this assumption on abductive part. The use of the multiple-faults assumption is useful if one is using multiple-context causal analysis within the reasoning engine [deKleer,87]. Our system uses a single-context based justification reasoning maintenance system [Doyle,78].

The use of abductive rules plays an important role in dealing with combinatorial explosion. The rules eliminate a significant number of possibilities by imposing a chosen state of a component based on heuristics. By using the rules, we effectively restrict the search space of diagnosis reasoning.

Chapter 4
System Implementation

We have developed a prototype of a diagnosis system which provides recommendations for mechanics to repair vehicle-engine-related problems. In this stage the engine system is comprised of the 12-volt power supply and the CCP systems. The diagnostic system requires a set of input from the hardware interface. We gather input data into a data file. Our diagnostic system can detects more than one causes of the failure. The output is a list of the abnormal components causing the symptoms observed. The diagnosis system also gives a list of the next measurements to perform in order to further localize the problem. The development of the diagnostic system for diagnosing vehicle engine's problem is a big task. Our system is the initial step that can be expanded in order to develop a complete diagnostic system dealing with the vehicle engine as the whole.

4.1. Vehicle Engine Organization

A vehicle engine is a very complicated system that needs a vast amount of knowledge to repair it. In figure 4-1, we show a "part of" relationship hierarchy for a vehicle engine system. We organize the vehicle engine system by classifying its components into three large categories; electronic control, engine and 12-volt power system.

Although the engine operation depends strongly on the work of the electronic control module (ECM), we classify the electronic control system that includes the ECM into a separate system. We do so because the electronic control system is a broader system controlling many vehicle functions besides the engine. The same reason is applied to the power supply system. Every subsystem consists of a number of subsystems or primitive components. The charcoal canister purge (CCP) system, a part of the emissions system, includes primitive components such as: hoses, wires, purge solenoid, charcoal canister and pressure control valve. This organization is not intended to show physical connections among subsystems or components, instead it illustrates a higher-level view of the physical components based on vehicle engine system functions.

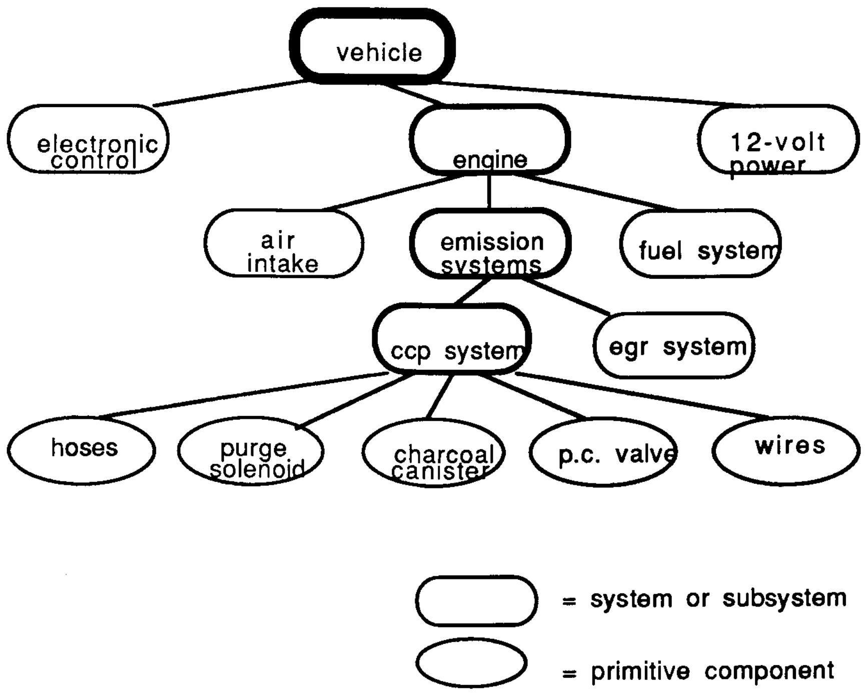

Figure 4-1
Composition Hierarchy

4.2. Vehicle Engine Model

4.2.1. Primitive Components

Engine behaviour is represented by the behaviour of the primitive components comprising the whole engine. Every replaceable physical component is considered to be a primitive component. Each primitive component has an input and/or output description as shown at figure 4-2.

A primitive component has one or more input-output terminals where input-output relations or correlations derived from physical laws are applied. For example, the behaviour of the battery can be observed from values of voltage going out and the behaviour of a wire can be seen from voltage and current coming in and going out through its terminals.

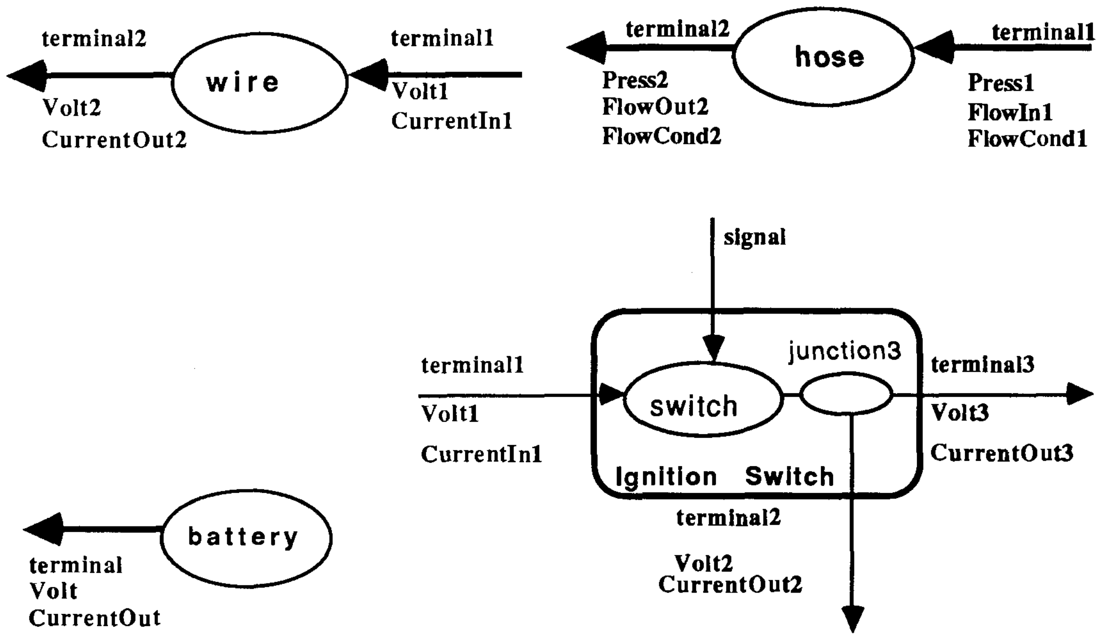

Figure 4-2
Primitive Components

The examples below are behaviour that can be observed on Battery and Wire components.

Battery :
(regarding to voltage)

$\text{volt} < 9$, or
$\text{volt} > 16$, or
$9 <= \text{volt} >= 16$.

Wire :
(regarding to current)

currentIn = 0 and currentOut = 0, or
currentIn = currentOut, or
currentIn =\= current Out.

We describe primitive components' behaviour based on the behaviour of input-output observed in its terminals. Behavioural description is adequate, meaning that it covers all possible behaviour at any of components' states or modes.

4.2.2. Physical Connectivity

Most of the connection links between components are based on equality, disequality or inequality. The use of mathematical functions is also possible. We use unification to establish input-output relationship among the components. Figure 4-3 illustrates the Electrical Power System's components' "part of" relationships hierarchy and figure 4-4 shows the physical connectivity among its components.

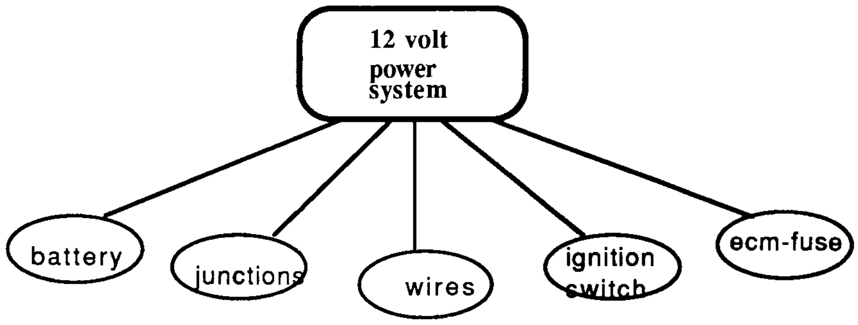

Figure 4-3
Power System Components

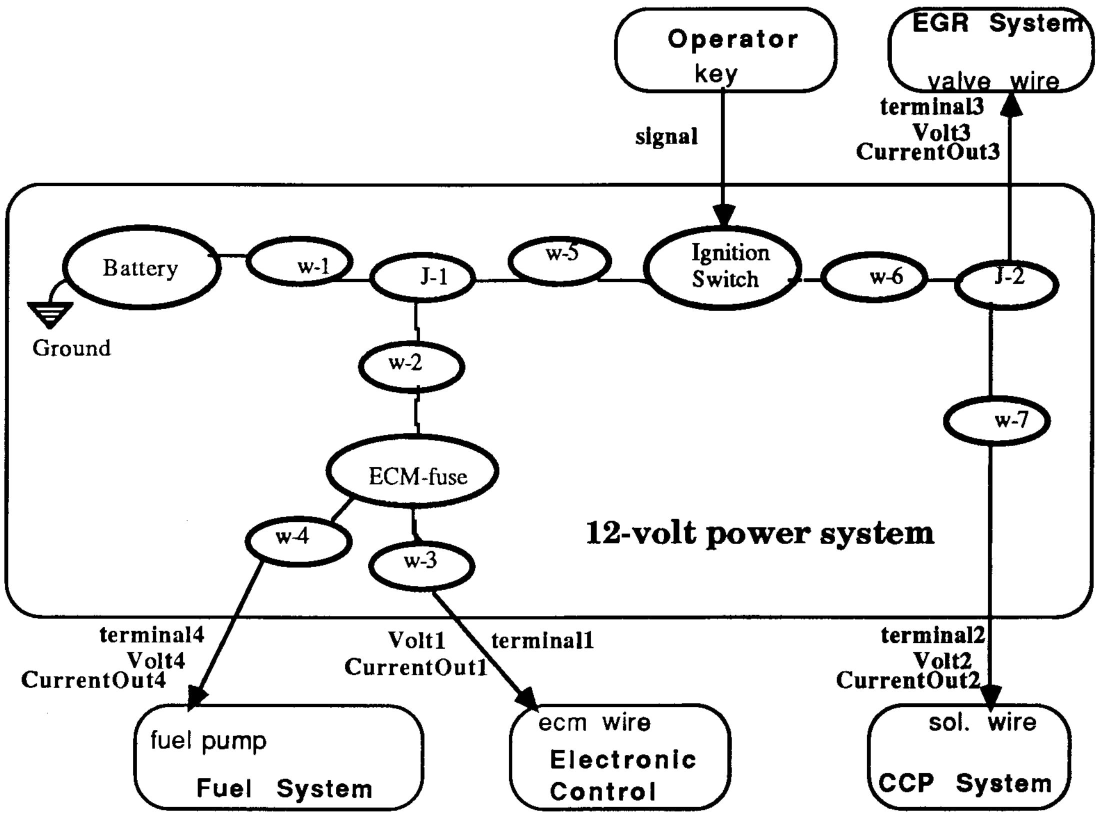

Figure 4-4
Power System's Physical Connectivity

The 12-volt power system includes a battery, wires, junctions, ignition switch and ECM-fuse. Every component is connected to adjacent components constructing a subsystem of the engine system. Physical parameter values are propagated through connections along all components according to design descriptions. The 12-volt of battery voltage is propagated to wire-1, junction-1, wire-5, ignition switch and so on, through unification.

All constraints are propagated through unifying values in two directions. The following are the examples of constraints manifest in the model :

- voltOut(Battery) = voltIn(Wire-1),
  currentOut(Battery) = currentIn(Wire-1).

- current(Wire-1) = currentIn(Junction-1),
  voltOut(Wire-1) = voltIn(Junction-1).

- flowIn(Injectors) = flowOut1(Injectors) +
  flowOut2(Injectors) +
  flowOut3(Injectors).

- pressureOut1(Pump) = 250 + pressureOut2(Pump).

A subsystem class is a composite of primitive components that embodies all constraints, parameters and methods. For example, the parameter values observed at terminal-2 of 12-Volt Power System are merely those of Wire-7.

currentOut2(Power_System) = currrentOut(Wire-7),
voltOut2(Power_System) = voltOut(Wire-7).

4.3. Knowledge Base

Knowledge of the diagnostic system is represented in Echidna knowledge base. A knowledge base consists of a set of schemata. Each schema is a model of a object class of vehicle's component. Figure 4-5 is to illustrate the system's knowledge base. A comment begins with the percent (%) sign.

```
conditionType = {good,bad}.
signalType = {off,on}.
stateRange = {0..9}.
voltageRange = {0..24}.
currentRange = {-20..20}.
pressRange = {0..104}.
flowRateRange =  {0..60}.

schema component
{
% type declarations :
        conditionType Condition.
        stateRange State.
        Name.

% methods :
        condition(Condition).
        state(State).
        name(Name).
        mode.
        build.
}


schema composite:component
{
% type declarations
component Clist.

% methods
        build:- buildSubComps(Clist).

buildSubComps([]).
buildSubComps([component Hcomp|Tcomps]):-
        Hcomp:build,
        buildSubComps(Tcomps).
}
```

Figure 4-5
Schema of Object Classes

All variables are bound to a specified domain over which the variable ranges. A schema 'component' represents an object class; this class is a top class of knowledge-base's schemata. A schema 'composite' is the second level of object class which inherits methods from the component object. Echidna supports only a single inheritance hierarchy where each class has exactly one superclass.

4.3.1. Classes and Instances

A schema can be instantiated to create one or more copies of an object class. This instance is modified through unification [Sidebottom,91b]. A new instance is created whenever a variable is declared to be associated to a particular class. For example, "IS isa igswitch" at the "build" method in figure 4-6. IS is an instance variable, it stores information about the state of the instance. An instance variable is persistent so that it can not be changed.

```
schema power:composite
{
% type declarations :
     igswitch IS. battery Battery.
          ----
          ----
     signalType Signal.

% accessors for persistent variables
      terminal1(Volt1,CurrentOut1).
          ----
          ----
     signal(Signal).

     order mode.

     mode:-
       IS:condition(good),
       Battery:condition(good),
       W1:condition(good),
       W2:condition(good),
       W6:condition(good),
       ECM_F:condition(good),
       Engine_F:condition(good),
       J1:condition(good),
      J2:condition(good),
      J3:condition(good),
      condition(good).

    mode:-
         condition(bad).

% define components of system

    build:-
         IS isa igswitch,       IS:name(ignitionswitch),
         Battery isa battery,   Battery:name(battery),
         W1 isa wire,           W1:name(wire1),
         W2 isa wire,           W2:name(wire2),
         W6 isa wire,           W6:name(wire6),
         J1 isa junction3,      J1:name(junction1),
         J2 isa junction3,      J2:name(junction2),
         J3 isa junction3,      J3:name(junction3),
         ECM_F isa fuse,        ECM_F:name(ecmfuse),
         Engine_F isa fuse,     Engine_F:name(enginefuse),

         Clist = [IS,Battery,W1,W2,W6,J1,J2,J3,ECM_F,Engine_F],

% input/output parameters of system

          J1:terminal2(Volt1,currentRange I1), 0-CurrentOut1 =:= I1,
          J1:terminal3(Volt2,currentRange I2), 0-CurrentOut2 =:= I2,

          J2:terminal2(Volt3,currentRange I3), 0-CurrentOut3 =:= I3,
          J2:terminal3(Volt4,currentRange I4), 0-CurrentOut4 =:= I4,

          J3:terminal3(Volt5,currentRange I5), 0-CurrentOut5 =:= I5,

         IS:signal(Signal),

% connections between components within system

           Battery:terminal(VoltA,CurrentA),
          J3:terminal1(VoltA,CurrentA),

          J3:terminal2(VoltB,currentRange CurrentB),
          W1:terminal1(VoltB,currentRange CurrentBB),
         0-CurrentB =:= CurrentBB,

          W1:terminal2(VoltC,CurrentC),
          IS:terminal1(VoltC,CurrentC),

          IS:terminal2(VoltD,CurrentD),
          W2:terminal1(VoltD,CurrentD),

          IS:terminal3(VoltE,CurrentE),
          W6:terminal1(VoltE,CurrentE),

          W6:terminal2(VoltF,CurrentF),
           ECM_F:terminal1(VoltF,CurrentF),

          W2:terminal2(VoltG,CurrentG),
          Engine_F:terminal1(VoltG,CurrentG),
        Engine_F:terminal2(VoltH,CurrentH),
        J1:terminal1(VoltH,CurrentH),

        ECM_F:terminal2(VoltI,CurrentI),
        J2:terminal1(VoltI,CurrentI),

       composite::build.

}
```

Figure 4-6
Basic Schema of Power System

As we see in the power system schema above, a subsystem is comprised of primitive components. Each of them is represented as a member of the class. The predicate 'build' is meant to define the components of the 12-volt power supply system. Classes are accessed by logical messages or unifying goals which match logical methods or predicates defined within schemata. Knowledge about the class is modular and localized within the schema. We create instances of schemata to represent real world objects. All components which comprise the power system are constructed when the goal "build" is issued.

Each parameter must have an associated domain of possible values declared and every instance of the class shares the same parameters. Schema instances can be sent messages, passed arguments, or composed into networks. All of these operations are performed through unification. Constraints are established by exchanging messages between a sending and receiving schema and the parameters of any of its components.

4.3.2. Methods and Logical Messages

There is a method called accessor since it simply provides an access to the instances variable, for example : signal (Signal). We can use an accessor to retrieve and set up the value of a parameter. The schema uses methods to maintain system's knowledge bases. The method library is built to retrieve the candidate clauses. The method lookup is used to resolve which one will be chosen. These all are performed at the compile time, therefore the set of methods is static during system executions. If the corresponding method is not defined, it will cause the error for the message sent. We can also have global methods which are defined at the outside of any schema definition in the knowledge base.

We use message-passing and message-interpretation for the communication between two schemas. A goal is a message sent to an object. A message is the name of the corresponding method that may be followed by arguments. Message interpretation is the selection of clauses from the method.

The logical message is the goal initiated by a sending schema. This is performed by unification of a logical parameters whose values may be refined towards the ground values. The process is monotonic and only reversible by dependency backtracking. A logical method is a logical predicate defined in the receiving schema. Unification is carried out in a non-deterministic way.

```
observation(Signal, V1, V2, V3, M3, M4):-
        default,
        apply_data(Signal, V1, V2, V3, M3, M4),
        generate_diagnoses.
Power:observation(on, 0.0, 0.0, 11.9, 11.9, 0.0).
```

Figure 4-7
Method and Logical Message

Figure 4-7 illustrates an example of a logical message sent into power system schema. The message "Power:observation(on, 0.0, 0.0, 11.9, 11.9, 0.0)" is issued and will be unified with methods defined in Power. Power is a schema variable which represents the 12-volt power supply system class.

4.3.3. Constraints and Schema Elaboration

Constraints support information flow through the link of persistent data. The engine constraint network is constructed during a session with Echidna. Constraints are usefull to prune the space before a search begins. There are two ways of constraints set up. The first one is by issuing the goal which applies constraint operator or a constraint relation. The second one is unification of two terms to be the same.

Constraint propagation is applied over variables which have an explicitly declared domain. A schema is a parametric model for a class. Instantiating the parameter of a schema to actual values specifies a possible member of the class. A particular instantiation is valid if it holds for all constraints which have been asserted within the instance. Constraint propagation causes the refinement of parameter domain.

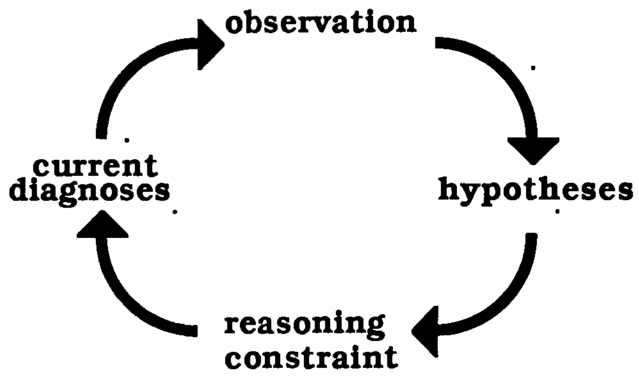

Figure 4-8[1]
Diagnosis Cycle

The instantiation process is incremental and employs hypothetical reasoning. Figure 4-8 illustrates an iterative process of making hypotheses during the diagnosis process. We use hypotheses to derive new constraints

[1]This is adopted from [Havens,83]

and apply the constraints to the schemas in the network. This reduces the search space. Some hypotheses might be retracted when it's deduced consequences do not match the observational data. In this case the process continues forward.

The reasoning process continues to refine models until they are fully ground. The process relies on applying constraints to the domains of parameter values. The process is incremental. Model-based diagnosis elaboration is carried out by applying internal heuristics to refining the model. Each new choice imposes new constraints on the network. Choices are made incrementally and their constraints propagated before any other hypothesis is pursued. Incremental elaboration allows flexible control structures and allows arbitrary heuristic order.

4.4. Deductive Model

A component model represents an object with some parameters corresponding to some physical or electrical units applicable to the object. It also describes the modes of components together with behavioral descriptions related to each of them. For example, component modes for 'Hose' could be: good, leaking, blocked, etc. Assuming there is liquid flowing through the hose, leaking hose means 'total amount of flow going out the hose is less than that of flow coming into the hose'. Each and every primitive component has its own possible modes and one mode can have more than one associated behaviour descriptions. The following is a model for the hose, it inherits all the methods defined in a schema component.

```
flowRange = {0..60}.
pressureRange = {0..104}.
conditionType = {good,bad}.
schema hose:component
{
% type declarations
pressRange Press1. flowRateRange FlowIn1. conditionType FlowCond1.
pressRange Press2. flowRateRange FlowOut2. conditionType FlowCond2.

% accessors for persistant variables
        terminal1(Press1,FlowIn1,FlowCond1).
        terminal2(Press2,FlowOut2,FlowCond2).
        press1(Press1). flowOut1(FlowIn1). flowCond1(FlowCond1).
        press2(Press2). flowOut2(FlowOut2). flowCond2(FlowCond2).

% modes of operation
order mode.
mode:- % good state
        FlowIn1 =:= FlowOut2,
        Press1 =:= Press2,
        FlowCond1 =:= FlowCond2,
        State =:= 0,
        Condition =:= good.

mode:- % leaking state
        FlowOut2 =\= FlowIn1,
        Press1 =:= Press2,
        State =:= 1,
        Condition =:= bad.

mode:- % blocked state
        FlowOut2  =:= 0,
        FlowIn1 =:= 0,
        State =:= 2,
        Condition =:= bad.
}
```

The model of a composite object is an engine theory that describes the component and its behaviour. It also describes the interrelationships of the components which make up the whole network structure. The structure of the complex object model implies the composition hierarchy of the whole engine consists of subsystems and primitive components. The previous figure 4-6 shows the model of Power System as a composite system. It describes the components comprising the system and its physical structure.

The following is the schema model of the vehicle system which consists of the 12-volt power supply and CCP system.

```
schema vehicle:composite
{
% type declarations :
        ccp CCP.
        power Power.

    voltageRange Volt1.   currentRange Current1.
    voltageRange Volt2.   currentRange Current2.

    order mode.
    mode:-
        Power:condition(good),
        CCP:condition(good),
        condition(good).
    mode:- condition(bad).

% system components :
    build:-
        Power isa power, Power:name(power_system),
        CCP isa ccp,    CCP:name(ccp_system),

        Clist = [Power,CCP],

% connections between components within system :

          Power:terminal2(VoltA,CurrentA),
          CCP:terminal2(VoltA,CurrentA),
         composite::build.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                          Abductive Part :                         %%
% containing rules at level of Vehicle

     owner_complaint(U1):- Power:owner_complaint(U1).

    observation_1(Signal,V1,V2,V3,C1,C2):-
         apply_data0(Signal,V1,V2,V3,C1,C2),
        generate_diagnoses.

    observation_1(Signal,V1,V2,V3,M4,C1,C2):-
         apply_data0(Signal,V1,V2,V3,M4,C1,C2),
        generate_diagnoses.

    observation_1(Signal,V1,V2,V3,M3,M4,C1,C2):-
        default,
         apply_data0(Signal,V1,V2,V3,M3,M4,C1,C2),
        generate_diagnoses.

     observation_2(M4):- Power:observation_2(M4).

    default:-
        Power:mode,
        CCP:mode.

    apply_data0(Signal,V1,V2,V3,C1,C2):-
         Power:observation_1(Signal,V1,V2,V3),
        CCP:m0(C1,C2).

    apply_data0(Signal,V1,V2,V3,M4,C1,C2):-
         Power:observation_1(Signal,V1,V2,V3,M4),
        CCP:m0(C1,C2).

    apply_data0(Signal,V1,V2,V3,M3,M4,C1,C2):-
         Power:observation_1(Signal,V1,V2,V3,M3,M4),
        CCP:m0(C1,C2).

    generate_diagnoses:-
          solution_list(Clist,[],PrintList),
         print(PrintList).

     solution_list([],PrintList,PrintList).
     solution_list([component Hcomp|Tcomps],PrintList,List1):-
```

```
        Hcomp:condition(Ccond),
        Hcomp:state(Cstate),
        Hcomp:name(Cname),
        solution_list(Tcomps,[[Cname,Ccond]|PrintList],List1).

}
vehicle Vehi.
```

We integrate physical organization and behavioral description in a deductive model. The model behaviour is determined by the behaviour of its components. In this way we overcome the difficulties of coordinating physical organization and behavioural model faced by [Hamscher,90a]. The model is adequate and guarantees a match between observation and at least one of the behaviour description that can be generated by the model.

4.5. Input Data

McCar-EAS [Joseph,89] is a tool already developed to gather data from the artifact. This system operates on a standalone PC Micro Computer for on-road data collection. It is an hardware interface consisting of an MC-68HC11 micro processor which automatically records engine physical measurements into data files, there are data such as temperature, voltage, and so on. Some examples of input data are described at section 3.2.1.

1. Long cranking time.
2. No start.
3. Hard start.
4. Poor performance.
5. Dieselling.
6. Excessive odor.

Figure 4-9
Some Common Drivability Complaints

Beside of measurements taken from McCar-EAS, the diagnosis system accepts drivability complaints from mechanics or owners in the garage.

This information drives the diagnosis process. Some commonly found drivability complaints are listed at figure 4-9. The drivability complaint could appears alone or together with others. We may consider some steps before starting diagnosis system such as visual or physical inspection, diagnostic circuit checks, etc. These are all related to some causes that are not covered in ECM-recorded information.

4.5. Diagnosis Example

We assume the vehicle engine structure in this example consists of only the 12-volt power supply system and the charcoal canister purge system (CCP). Each of the power supply system and CCP is a composite system that has several primitive components at the lower level within the compositional hierarchy. V is a schema variable of the simplified vehicle structure. The first time the system starts, a logical message is sent into V. The message must be sent to the particular schema variable, for example "V isa vehicle". This is to declare V to be associated to the vehicle schema class. "V:build" is the message that will be unified with a method defined in the vehicle schema. Building the structure is performed in a cascade from the 'top' level object downward through the compositional hierarchy.

The diagnosis process is initiated by sending the message containing input or observational data, e.g.:

V:observation(Signal, V1, V2, V3, M3, M4, C1, C2).

This data will be sent to the appropriate subsystems where the underlying physical unit checkpoints taken. In this example, there are six data observed at power system components' terminal and the rest two are taken at the CCP system. These data are applied into the structure and become persistent. We concentrate on the 12-volt power supply system to explain diagnosis examples. Figure 4-10 shows some test points of the 12-volt power supply system.

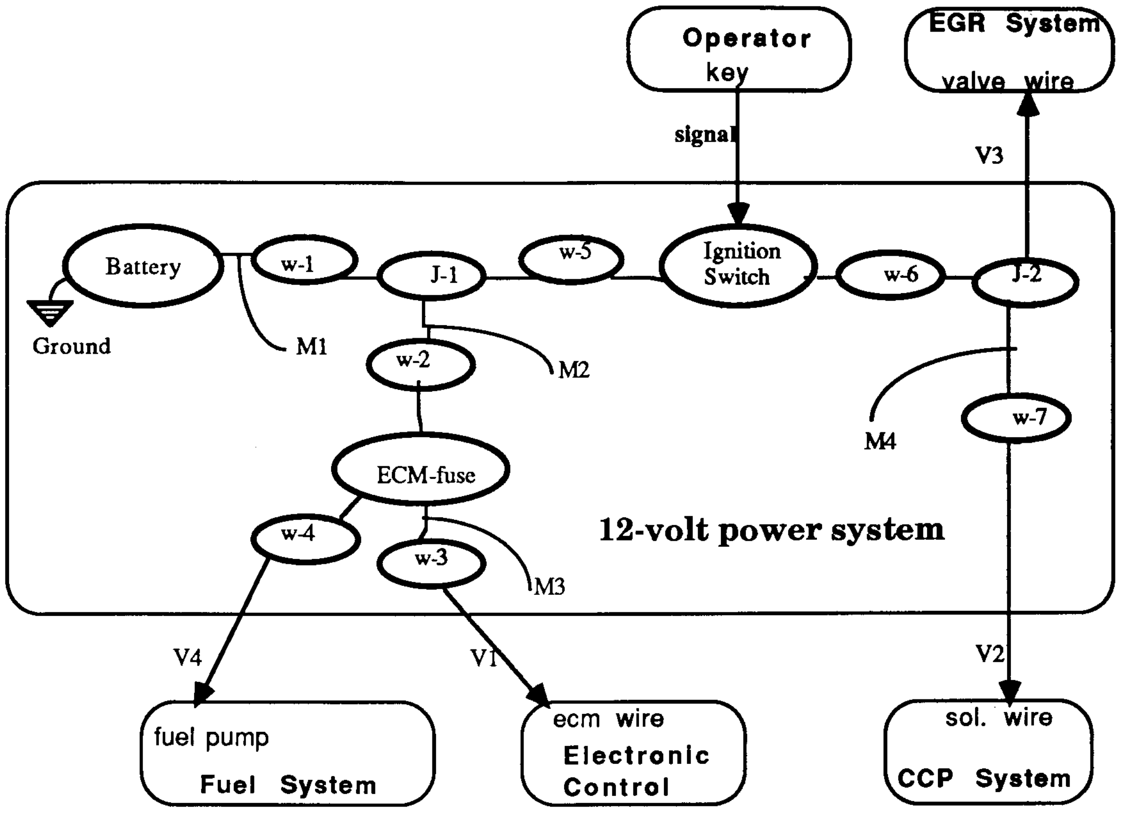

Figure 4-10
12-Volt Power Supply System

The data should be firstly applied into the model for diagnosing faulty components, because we want to refine and check the consistency of the model against the actual data. The model may be inconsistent with the actual behaviour and this drives the diagnoses generation. Diagnosis is a the process of refining the model until it matches the observations.

The system uses abductive rules heuristically and always assumes the engine to be initially in the good state. The use of abductive rules can be seen in the power schema class at figure 4-11 below. After the above logical message sent, the corresponding method in vehicle schema will be selected. This new observational data initiates diagnosis process which put the whole diagnosed engine at the default states (good states). The method "apply_data0" applies the observational data into the model in order to diagnose the faulty components. If the good state of engine is still consistent, the diagnosis is over. Otherwise the system uses abductive rules by issuing the message "apply_rule0". The *a priori* probability method applies when rules are exhausted.

We select the next probe to localize the faulty components. Selecting the next probe is part of the abductive process and it is based on heuristics described by shallow knowledge of the system.

```
schema power:composite
{

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                       Abductive Part :                            %%
    owner_complaint(dieselling):-
             ECM_F:condition(bad).

    owner_complaint(hard_start):-
             IS:condition(bad).

    observation_1(Signal,V1,V2,V3):-
             apply_data0(Signal,V1,V2,V3),
             generate_diagnoses.

    observation_1(Signal,V1,V2,V3,M4):-
             apply_data0(Signal,V1,V2,V3,M4),
             generate_diagnoses.

    observation_1(Signal,V1,V2,V3,M3,M4):-
             default,
             apply_data0(Signal,V1,V2,V3,M3,M4),
             generate_diagnoses.

    observation_2(M4):- apply_data2(M4),
               generate_diagnoses.

    default:- print(accessing_deductive_model_power),
         IS:mode,
         Battery:mode,
         W1:mode,
         W2:mode,
         W3:mode,
         W4:mode,
         W5:mode,
         W6:mode,
         W7:mode,
         ECM_F:mode,
         J1:mode,
         J2:mode.

    apply_data0(Signal,V1,V2,V3):-
            signal(Signal),
        volt1(V1),
        volt2(V2),
        volt3(V3).

 apply_data0(Signal,V1,V2,V3,M4):-
        signal(Signal),
        volt1(V1),
        volt2(V2),
        volt3(V3),
         J2:terminal2(M4,MM4).

 apply_data0(Signal,V1,V2,V3,M3,M4):-
        signal(Signal),
        volt1(V1),
        volt2(V2),
        volt3(V3),
         ECM_F:terminal2(M3,MM3),
         J2:terminal2(M4,MM4),
         apply_rule0(Signal,V1,V2,V3,M3,M4).
       a_priori.

 apply_data2(M4):-
     J2:terminal2(M4,MM4),
     apply_rule2(M4).

 apply_rule0(Signal,V1,V2,V3,M3,M4):- print(rule1),
        Signal =:= on,
       V2 > 10,
       V1 < 5,
       M3 > 10,
         ECM_F:condition(bad).

 apply_rule0(Signal,V1,V2,V3,M3,M4):- print(rule1),
        Signal =:= on,
       V1 < 5,
       V2 < 5,
       V3 > 10,
        W7:condition(bad).

 apply_rule0(Signal,V1,V2,V3,M3,M4):- print(rule1),
        Signal =:= on,
       V1 < 5,
       V2 < 5,
       V3 > 10,
        W1:state(2).

 apply_rule0(Signal,V1,V2,V3,M3,M4):- print(rule1),
        Signal =:= on,
       V1 > 10,
       V2 < 5,
        J2:condition(bad).

apply_rule2(M4):-
       M4 > 10,
        J1:condition(good).

a_priori:- W7:condition(bad).
a_priori:- Battery:condition(bad).
```

```
    a_priori:- IS:condition(bad).
    a_priori:- J2:condition(bad).

    measure1(C1,C2):-
    print(rule0_4),
          apply_data1(C1,C2),
          apply_rule1.

    apply_data1(C1,C2):-
         currentOut1(C1),
         currentOut5(C2).

    apply_rule1:-
       C1 > 10,
         J1:condition(good).

    apply_rule1:-
       C1 > 0,
         J1:condition(bad).

    apply_rule1:-
    print(apply_apriori),
        a_priori_2.

    a_priori_2:- Battery:state(1).
    a_priori_2:- IS:condition(bad).
    a_priori_2:- ECM_F:condition(bad).

    generate_diagnoses:-
          solution_list(Clist,[],PrintList),
         print(PrintList).

      solution_list([],PrintList,PrintList).
     solution_list([component Hcomp|Tcomps],PrintList,List1):-
         Hcomp:condition(Ccond),
         Hcomp:state(Cstate),
         Hcomp:name(Cname),
          solution_list(Tcomps,[[Cname,Cstate]|PrintList],List1).

}
```

Figure 4-11
Abductive Part of The Power System Schema

We simulate diagnosis process by sending :
"Power:observation(on, 0.0, 0.0, 11.9).",
These are the data about voltages on all output terminals of the power system, the signal coming from the key on/off, and the voltage from V1, V2, and V3 consecutively. The system puts the engine structure at state good and this will checked whether it they are consistent to the data or not.

Should any discrepancy occurs, the rules, *a priori* probability method and the deductive process applies in order. In this particular example the match between data and model results in two faulty components: Junction-1 is open on terminal 1 and Junction-2 is open on terminal 3. Some of diagnosis results is shown in the table 4-1. Those are based only the deductive process without intervention of the abductive part.

12-Volt Power Supply System										
#	Signal	V1	V2	V3	M1	M2	M3	M4	Faulty Component(s)
1	off	11.9	0.0						All components are good
2	off	0.0	0.0						Fuse: broken.
3	on	0.0	0.0	11.9					J-1:open; J-2:open.
4	on	0.0	0.0	11.9				11.9	J-1:open; J-2:open; W-7:open.
5	on	0.0	0.0	11.9			0.0	11.9	W-2:open; W-7:open.

Table 4-1
Diagnosis Results

On the example #3 of table 4-1, we see that the result is not plausible to occur in reality. The system hypothesizes J-1 to be open on the terminal-1 close to the battery. Therefore there is no voltage going out of output terminals. Since the voltage at terminal-3 of 12-volt power supply system (V3) is observed to be 11.9 volt, the system assumes that J-2 to be broken because it produces 11.9 volt at the output terminal although there is no incoming voltage. The diagnosis proceeds by taking another measurement at M4 (M4 = 11.9 volt). Even worse the system falsifies W-7 because it produces V = 0.0 after receiving 11.9 volt coming from J-2.

The diagnosis becomes reasonable when we apply the knowledge from diagnostic procedure based on mechanic's experience. Given the observational data of {Signal = on; V1 = 0.0; V2 = 0.0; V3 = 11.9; M4 = 11.9}, it activates a rule in the system knowledge base: "if M4 > 10.0 then J-1: condition (good)". This rule applies and yields ECM-F:condition(blown) and W-7:condition(open) as the faulty components which are consistent with the observations. The car owner's complaint is used in abductive part for guessing the faulty component. Having the same observation above {Signal = on; V1 = 0.0; V2 = 0.0; V3 = 11.9; M4 = 11.9} and the complaint from the car owner: "dieselling", the diagnostic system recommends ECM-F:condition(bad) and W-7:condition(open). The knowledge of diagnostic procedures is incorporated in the abductive part of system knowledge bases.

Chapter 5
System Evaluation

This thesis has presented a model-based diagnosis system. We have developed a system design which combines abductive and deductive diagnosis. We have successfully integrated model-based diagnosis with the use of heuristic knowledge. In this chapter, we examine the implementation of the diagnostic system and examine its design. We also discuss the applicability of system design to real data, problems encountered and a brief comparison with other systems found in the literature.

5.1. Design Analysis

5.1.1. Deductive Model Issues

The use of deductive models is the key idea in this vehicle engine diagnostic system. Models of component objects and generative function states give great flexibility in adding, subtracting, modifying or replacing the models according to the ones desired. Building a deductive model is a continuous process and the model typically needs to be refined as the system is used and tested.

The deductive model of a component is still very simple compared to the actual component. The model is adequate but may not be ideally competent. For example, to characterize a wire based on the voltage and current coming and going from its terminal may be not sufficient. There are other parameters such as impedance, grounding state or life-time factors which affect wire behaviour. In fact the ideal competent model is an object in the real world.

We only represent the failures which are worth representing in our model [Hamscher,90b]. It is impossible to capture every detail of an actual device in the model. For example, a mechanic usually replaces the whole of a cable including a number of wires when it fails, although only one of them may cause the failure. We can control this problem by separating the distinguishable failures. If there are many faults with the same repair, we treat them as one fault.

Modelling of predicted behaviour still has limitations. The ideal system would capture transient events, but this is too expensive and is not available in our system. We only predict behaviour over a stable long interval, although this simplification seems to be reasonable so far to provide correct diagnoses. We also use a level-based behavioural model. Behaviour is represented at the level of components which can be replaced by mechanics and modeled by a knowledge engineers. We do not model the behaviour of electrons in a wire, instead we represent the conditions of electrical current or voltage.

We represent the physical organization of the artifact using primitive components. Components must be represented in terms of observable attributes and behavioural abstractions. The representation should correspond to the possible failures of actual devices. On the other hand, we only model failure modes for the components that have a high likelihood of failure. We can not model failures which give rise to very complex behaviour, but we will model a failure mode if it is simple enough to model [Hamscher,90b]. There are encapsulated components that must be treated as one component, although they consist of more than one primitive component.

In the diagnosis process, the model does not necessarily know the input-output 'values' coming and going between components at a particular state as required by Struss [89]. *GDE+* of [Struss,89] actually uses this knowledge to exploit the contradiction between correct behavior and observation. In contrast, our deductive model finds discrepancy based on the behavioural description of each component and backtracks immediately to adjust the whole structure. The deductive model enables the system to find malfunctioning components whenever an observation of the real world is provided, therefore the technique of Struss[89] to confirm malfunctioning components by observation is not necessary. Our system perform the next probe if the current observations is not sufficient to recommend faulty components.

5.1.2. Observational Abstraction Issues

Vehicle engine diagnosis is a difficult problem. It deals with a very complex system. The system consists of components which has theories of physics determining the input-output relationships among them. Representation of the physical organization of components in a deductive model requires corresponding information from a real-time data-acquisition tool. The data available in the current system are adequate, but could be improved to meet observation requirements. Most of them are already in compiled form and more appropriate for a rule-based expert system input data. This is because the tool was designed for a rule-based type of diagnosis system. However, modification of the data acquisition tool in the near future should be straightforward.

The measurement of data is considered much more expensive than the computational cost of selecting them. It would cost more if we performed observations over a long interval of time, therefore temporal abstraction in observation is useful. The observation tool has limitations that may make the system unable to localize faults. Sometimes it is too expensive to make the precise observation that is needed to discriminate diagnoses. For example, the pressure at a hose being 80 kPa or 100 kPa may cause significantly different results in diagnoses, but this kind of precision may not be economically feasible. Typically, the available data or mechanics only know whether there is 'enough' pressure or not at a particular hose. The diagnostic system requires input from a real time data acquisition tool. The available data in the current system do not provide all the information needed. Some data fulfill the requirements indirectly so that transformation processes are needed.

5.1.3. Abductive Process Issues

The use of heuristics together with model-based diagnosis improves system performance and gives a procedural meaning to diagnosis. For a simple small system with few components, the use of rules seems to cover almost all possibilities of diagnosis. But, for complex and big system, which is the general case, it is highly improbable to get a complete coverage in the abduction part by using heuristics. The power of a deductive model could be minimal for small problems but would increase with system complexity.

Diagnoses = Wire-2:open; Wire-7:open

Combination of Deductive and Abductive Part Performance	Average Performance (ΔT)
Deductive Process Only	99 seconds
Abductive Rules Applied	96 seconds
A priori Probabilty Applied	96 seconds
Abductive Part + Deductive Process	108 seconds

Table 5-1
System Performances

The *a priori* probability method also contributes to improving performance. This method is a simple method that relies on the single-fault assumption. The single-fault assumption decreases the number of possible diagnoses. Table 5-1 shows the performance of the diagnostic system with the uses of a pure deductive process and abduction part. The abductive part is the use of rules and *a priori* likelihood of failures. We expect that the rules are almost always correct. The use of rules increases the speed of diagnosis and so does the *a priori* likelihood method. However, if the abductive part fails, meaning that the rule applied is not consistent with observations, it results in more time needed for the system to find diagnoses.

Moreover, the performance shown on table 5-1 does not seem very impressive. The current implementation of the diagnostic system does not elaborate the abductive rules efficiently. When any component is hypothesized to be in a different state, the Echidna reasoning engine backtracks immediately. This results in inefficiency because this backtrack may involve many choices at the rules and model made during the diagnosis. Unfortunately, we can not avoid this inefficient backtrack in the current implementation, because Echidna does not provide the proper tool to do so. Instead, we want to perform abductive rules by using the current state of the diagnosed object plus the new hypothesis asserted in the rules and then continue the diagnosis process. By doing so, we can maintain the incremental notion of the diagnostic procedure. In the future, we need the facility from Echidna to enable us to create a clone of the current diagnosed object which includes its entire deductive state. Using this clone, we can then perform efficient backtrack search of its state as necessary to perform diagnosis.

The use of the single-fault assumption in our system is not necessarily inferior to the multiple-faults assumption of [deKleer,89]. The single-fault assumption enables us to find solutions very efficiently, whereas the multiple-faults assumption which relies on multiple-context diagnosis reasoning tends to be very expensive. The latter approach has an expensive overhead in probabilistic computation required in finding global solutions. We use a less expensive single-context based diagnosis reasoning to produce one solution at one time. The cost of our method is to control and discriminate solutions sequentially. This approach is natural and reasonable, especially with the fact that the system always give the best predicted solution first.

5.2. Computational Issues

We use an adequate model to represent real objects. This model might maldiagnose when an important characterizing parameters is discarded. This could cause the system to assume the faulty engine to be in a normal state. Building an expert system is a continuous process and has to be refined if there is new knowledge regarding to the deductive model or abductive part. The domain expert who possesses a level of knowledge like a mechanic should be able to update the generative model of functional states or change the contents of the abductive rule-base.

The diagnosis process is performed in a monotonic fashion. Data measurement taken on physical organization checkpoints is carried out once and these data persist. Any new data supplied by the next probes selected are in addition to the data already given. It also means diagnosis refinement maintains monotonicity as well. The more data provided from the outside world, the more constraints are applied and the smaller the search space remaining in which to find hypotheses. Facts observed from the real engine are persistent whereas the hypotheses are reversible, based on dependency-directed backtracking.

The speed of computation is always an obstacle in diagnosis. Because of the search, the system produces many alternatives and grows exponentially according to the number of components in the engine. The diagnostic system reduces the search space in localizing faults by using constraints. This method together with the CLP constraint-solving-technique increases the speed significantly [Havens,90b].

Although the use of the single-fault assumption in abductive rules limits the system, it has been helpful to overcome the combinatorial problem in a diagnostic system. Yet, the use of this method still allows us to have complete and correct diagnoses. We only use the single-fault assumption in the abductive part, because it is impractical to enumerate all combinations based on multiple faults. It is important to note that the deductive model can capture multiple-faults, this may cause computational cost too.

5.3. Comparative Results

Any model-based diagnosis system that models only correct behaviour has limited performance, it loses additional diagnostic discrimination power and the unlikely faulty modes are considered along with the most likely ones. *Sherlock* [deKleer,89] uses behavioral modes which include correct and fault modes and assigns *a priori* likelihood. Unfortunately the use of behavioural modes results in a combinatorial problem as the number of alternatives increases exponentially. *Sherlock* uses probabilistic information to focus diagnoses. Like *sherlock*, our system uses behavioural modes in diagnosis. In contrast to *sherlock*, we incorporate the shallow knowledge including diagnosis procedure as parts of system's knowledge bases. This abductive part controls diagnosis sequences and heuristically suggests the next measurements required in discriminating diagnoses too. Another system that combines the shallow and deep knowledge is [Abu-Hanna,88].

Struss[89] controls diagnostic alternatives by concentrating on small candidate sets which are more likely to fail. It uses the single-fault assumption and argues that multiple faults can be treated as combinations of single faults. His system applies a control strategies to overcome combinatorial problem. It uses a method to confirm the correctness of components and rules out the implausible diagnosis based on a hypothetical value. Our system differs from [Struss,89] because our system's knowledge-base does not need to know something like what is the correct voltages in particular wire at any time. Instead, the model deduces it based on behavioural descriptions.

Knowledge about physical structure and a component's behaviour in carrying out its function is the crucial part of a model-based diagnostic system. Hamscher[90a] uses separated physical and functional organization in the knowledge base. This has caused technical problems in coordination. Our system is the first among model-based diagnostic systems which proposes an integrated model of fault and correct modes in a deductive model. There is no coordination problem in our system.

Our diagnostic system uses behavioural modes and a simple probability method. We use the single fault assumption in the probabilistic method for the sake of efficiency. Our system captures multiple faults as deKleer[89] does although we use a less expensive single-context reasoning diagnosis. We use rules and probabilistic method to cope with combinatorial problems and successfully combine model-based diagnosis with heuristics.

Chapter 6
Conclusion

The objectives of this thesis are to design a model-based diagnosis design and implement it in the Echidna constraint reasoning system. We have proposed an architecture for a diagnostic system which combines a deductive model and an abductive rule-base. The system has been implemented in prototype form that can demonstrate the power of the model-based approach in engine troubleshooting. We successfully built the deductive model for parts of engine system namely 12-volt power supply system and the CCP system. The model is sufficient to test our system, although further work is still needed to make it more realistic.

The work done through the completion of this thesis gives us experiences in using Echidna, a new CLP language, to solve the diagnosis problems. On the other hand, this work has been beneficial to push the research and development of the Echidna. We have explored the potential power of Echidna to enable a knowledge engineer to build a model-based diagnostic system. Echidna needs to improve its feature to enable the designer to make a copy of the complex object instance and provide the control mechanism in order to capture multiple solutions at one time. This feature is not available now, so that it restricts us to consider one solution at one time in solving diagnostic problem.

Model-based diagnosis approach enables us to build a diagnostic system for vehicle engine problems, the diagnostic system relies mainly on the knowledge from design descriptions. It gives us the way to avoid the necessity to enumerate exhaustively symptom-cause relationships. Diagnosis is driven by discrepancy between observation and predicted behaviour. In this system hypothesis generation is carried out based on knowledge of internal processes and components' interrelationships. This notion is very natural in diagnosis problem solving and ensures a high degree of confidence for the diagnostic system.

We integrate correct and fault modes in the deductive model. Reasoning in the deductive model is performed using a dependency-directed backtracking algorithm. This algorithm is important in our diagnosis system, it avoids thrashing behaviour. The use of heuristics for abduction increases the speed and efficiency. Heuristics play a beneficial role when the system faces obvious, well-known problems. The single-fault assumption used in our probabilistic method is also worth noting. This chosen strategy is a compromise solution for a system that doesn't consider multiple-context in finding a global solution.

The use of heuristic rules only for a diagnostic system is not adequate. Complete coverage is impossible to achieve. On the other hand, relying only on a deductive model is adequate, but it is inefficient due to the system's ignorance of the mechanic's experience in solving obvious problems. Combining both of them is the ideal solution. Our system successfully puts them together. We adopt the idea of differential diagnosis to discriminate diagnoses. In differential diagnosis, the system imposes structure to solve the problem. This technique considerably reduces search space.

The single-fault assumption may cause problems. DeKleer[89] has shown the advantages of using multiple faults in diagnosis reasoning. However, this approach also seems to have its own problem. The multiple-fault assumption requires a high overhead to consider all solutions at one time, that may be unnecessary. Study of performance comparison of these two approaches would be very useful.

This thesis demonstrates the beauty and potential power of the Echidna constraint reasoning engine as a new-generation Expert System Shell. It has a clean knowledge representation and supports constraint-based hypothetical reasoning elegantly.

This work suggests further studies to improve performance. Continued study on modelling of physical components is clearly needed to develop a viable diagnosis system for engine troubleshooting. Incorporating probabilistic information in the underlying system reasoning engine is also worth investigating. The results of this study would enable us to use multiple-context consideration applied at [deKleer,89]. It would be very interesting to know the performance of our diagnostic system using that approach. In the long run, adding the learning ability into the system could be beneficial, because it leads to a more efficient system which tackles the problems as quick as possible using a shallow level of knowledge.

Appendix

System Knowledge Base:

```
conditionType = {good,bad}.
signalType = {off,on}.

stateRange = {0..9}.
voltageRange = {0..24}.
currentRange = {-20..20}.
pressRange = {0..104}.
flowRateRange = {0..60}.


schema component
{

% type declarations :
     conditionType Condition.
     stateRange State.
    Name.

% methods :
     condition(Condition).
     state(State).
    name(Name).
    mode.
    build.
}


schema composite:component
{

% type declarations

component Clist.

% methods

      build:- buildSubComps(Clist).

     buildSubComps([]).

     buildSubComps([component Hcomp|Tcomps]):-
          Hcomp:build,
          buildSubComps(Tcomps).
```

```
%-----------------------------------------------
}


schema fuse:component
{

% type declarations

      voltageRange Volt1. currentRange CurrentIn1.
      voltageRange Volt2. currentRange CurrentOut2.

% accessors for persistant variables

       terminal1(Volt1,CurrentIn1).
       terminal2(Volt2,CurrentOut2).
       volt1(Volt1). currentIn1(CurrentIn1).
       volt2(Volt2). currentOut2(CurrentOut2).

% modes of operation
     order mode.
     mode:- % good state
           CurrentOut2 =:= CurrentIn1,
           Volt2 =:= Volt1,
          State =:= 0,
          Condition =:= good.

     mode:- % bad state - fuse blown
          CurrentOut2 =:= 0,
          CurrentIn1 =:= 0,
          State =:= 1,
          Condition =:= bad.
}


schema hose:component
{

% type declarations

      pressRange Press1. flowRateRange FlowIn1. conditionType FlowCond1.
      pressRange Press2. flowRateRange FlowOut2. conditionType FlowCond2.

% accessors for persistant variables

       terminal1(Press1,FlowIn1,FlowCond1).
       terminal2(Press2,FlowOut2,FlowCond2).

       press1(Press1). flowOut1(FlowIn1). flowCond1(FlowCond1).
       press2(Press2). flowOut2(FlowOut2). flowCond2(FlowCond2).

% modes of operation
     order mode.
     mode:- % good state
           FlowIn1 =:= FlowOut2,
          Press1 =:= Press2,
           FlowCond1 =:= FlowCond2,
            State =:= 0,
            Condition =:= good.

      mode:- % leaking state
             FlowOut2 =\= FlowIn1,
            Press1 =:= Press2,
            State =:= 1,
            Condition =:= bad.

      mode:- % blocked state
            FlowOut2 =:= 0,
            FlowIn1 =:= 0,
            State =:= 2,
            Condition =:= bad.
}


schema junction3:component
{

% type declarations

       voltageRange Volt1. currentRange CurrentIn1.
       voltageRange Volt2. currentRange CurrentIn2.
       voltageRange Volt3. currentRange CurrentIn3.

% accessors for persistant variables

        terminal1(Volt1,CurrentIn1).
        terminal2(Volt2,CurrentIn2).
        terminal3(Volt3,CurrentIn3).
         volt1(Volt1). currentIn1(CurrentIn1).
         volt2(Volt2). currentIn2(CurrentIn2).
         volt3(Volt3). currentIn3(CurrentIn3).

% modes of operation
           order mode.
      mode:-  % good condition
             Volt1 =:= Volt2,
             Volt2 =:= Volt3,
             CurrentIn1 + CurrentIn2 + CurrentIn3 =:= 0,
            State =:= 0,
             Condition =:= good.

      mode:-  % open circuit on 1
             CurrentIn2 + CurrentIn3 =:= 0,
            CurrentIn1 =:= 0,
             Volt2 =:= Volt3,
            State =:= 1,
            Condition =:= bad.

      mode:-  % open circuit on 2
             CurrentIn1 + CurrentIn3 =:= 0,
            CurrentIn2 =:= 0,
             Volt1 =:= Volt3,
            State =:= 2,
            Condition =:= bad.
```

```
     mode:- % open circuit on 3
          CurrentIn1 + CurrentIn2 =:= 0,
          CurrentIn3 =:= 0,
          Volt1 =:= Volt2,
          State =:= 3,
          Condition =:= bad.

     mode:- % shorted circuit
          Volt1 =:= Volt2,
          Volt2 =:= Volt3,
          CurrentIn1 + CurrentIn2 + CurrentIn3 =\= 0,
          State =:= 4,
          Condition =:= bad.
}


schema switch:component
{

% type declarations

      voltageRange Volt1. currentRange CurrentIn1.
      voltageRange Volt2. currentRange CurrentOut2.
     signalType Signal.

% accessors for persistant variables

      terminal1(Volt1,CurrentIn1).
      terminal2(Volt2,CurrentOut2).
      volt1(Volt1). currentIn1(CurrentIn1).
      volt2(Volt2). currentOut2(CurrentOut2).
     signal(Signal).

% modes of operation
     order mode.
     mode:- % Off condition
          Signal =:= off,
          CurrentIn1 =:= 0,
          CurrentOut2 =:= 0,
          State =:= 0,
          Condition =:= good.

     mode:- % On condition
          Signal =:= on,
          CurrentIn1 =:= CurrentOut2,
          Volt1 =:= Volt2,
          State =:= 1,
          Condition =:= good.

     mode:- % short circuit across throw
          CurrentIn1 =:= CurrentOut2,
          Volt1 =:= Volt2,
          State =:= 2,
          Condition =:= bad.

     mode:- % open circuit across throw
          CurrentIn1 =:= 0,
          CurrentOut2 =:= 0,
          State =:= 3,
          Condition =:= bad.
}


schema wire:component
{

% type declarations

     voltageRange Volt1. currentRange CurrentIn1.
     voltageRange Volt2. currentRange CurrentOut2.
      volt1(Volt1). currentIn1(CurrentIn1).
      volt2(Volt2). currentOut2(CurrentOut2).

% accessors for persistant variables

      terminal1(Volt1,CurrentIn1).
      terminal2(Volt2,CurrentOut2).

% modes of operation
     order mode.
     mode:- % good state
           Volt1 =:= Volt2,
           CurrentIn1 =:= CurrentOut2,
          State =:= 0,
           Condition =:= good.

     mode:- % shorted state
           Volt1 =:= Volt2,
           CurrentIn1 =\= CurrentOut2,
          State =:= 1,
           Condition =:= bad.

     mode:- % open state
           CurrentIn1 =:= 0,
           CurrentOut2 =:= 0,
          State =:= 2,
           Condition =:= bad.
}

schema canister:component
{

% type declarations

     pressRange Press1. flowRateRange FlowOut1. conditionType FlowCond1.
     pressRange Press2. flowRateRange FlowIn2. conditionType FlowCond2.
     pressRange Press3. flowRateRange FlowIn3. conditionType FlowCond3.

% accessors for persistant variables

      terminal1(Press1,FlowOut1,FlowCond1).
      terminal2(Press2,FlowIn2,FlowCond2).
      terminal3(Press3,FlowIn3,FlowCond3).

      press1(Press1). flowOut1(FlowOut1). flowCond1(FlowCond1).
      press2(Press2). flowOut2(FlowIn2). flowCond2(FlowCond2).
```

```
        press3(Press3). flowOut3(FlowIn3). flowCond3(FlowCond3).

% modes of operation

      order mode.

      mode:- % good state, storing but not purging
            FlowOut1 < 5,
            FlowIn2 < 5,
            FlowIn3 > 50,
            Press3 =:= Press1,
            State =:= 0,
            Condition =:= good.

      mode:- % good state, purging but not storing
            FlowIn3 < 5,
             FlowOut1 =:= FlowIn2,
            Press1 < Press2,
            Press1 =:= Press3,
            State =:= 1,
            Condition =:= good.

      mode:- % good state, not storing nor purging
            FlowOut1 < 5,
            FlowIn3 < 5,
            FlowIn2 < 5,
            Press1 =:= Press3,
            Press1 =:= Press2,
            State =:= 2,
            Condition =:= good.

      mode:- % bad state, blocked inlet no. 2
            FlowOut1 < 5,
            FlowIn2 < 5,
            State =:= 3,
            Condition =:= bad.

      mode:- % bad state, blocked inlet no. 3
            FlowIn3 < 5,
            State =:= 4,
            Condition =:= bad.

      mode:- % bad state, leaking
             FlowOut1 =\= FlowIn2,
            State =:= 5,
            Condition =:= bad.
}


schema pcv:component
{

% type declarations

       pressRange Press1. flowRateRange FlowOut1. conditionType FlowCond1.
       pressRange Press2. flowRateRange FlowIn2.  conditionType FlowCond2.
       pressRange Press3. flowRateRange FlowIn3.  conditionType FlowCond3.
```

```
% accessors for persistant variables

        terminal1(Press1,FlowOut1,FlowCond1).
        terminal2(Press2,FlowIn2,FlowCond2).
        terminal3(Press3,FlowIn3,FlowCond3).

         press1(Press1). flowOut1(FlowOut1). flowCond1(FlowCond1).
        press2(Press2). flowIn2(FlowIn2).   flowCond2(FlowCond2).
        press3(Press3). flowIn3(FlowIn3).   flowCond3(FlowCond3).

% modes of operation

      order mode.

      mode:- % Valve is closed by vacuum on terminal 1
           Press1 < 0,
           FlowOut1 < 5,
           FlowIn2 < 5,
           State =:= 0,
           Condition =:= good.

      mode:- % Valve is opened by pressure on terminal 2
           Press1 >= 0,
           Press2 > 80,
           Press1 =:= Press2,
           FlowOut1 =:= FlowIn2,
           FlowCond1 =:= FlowCond2,
           State =:= 1,
           Condition =:= good.

      mode:- % Valve is opened by vacuum on terminal 3
           Press1 >= 0,
           Press3 < 0,
           Press1 =:= Press2,
           FlowOut1 =:= FlowIn2,
           FlowCond1 =:= FlowCond2,
           State =:= 2,
           Condition =:= good.

      mode:- % Valve blocked
           FlowIn2 < 5,
           FlowOut1 < 5,
           State =:= 3,
           Condition =:= bad.

      mode:- % Valve stuck open
           Press1 =:= Press2,
           FlowOut1 =:= FlowIn2,
           State =:= 5,
           Condition =:= bad.

      mode:- % Valve restricted
           Press1 =\= Press2,
           FlowOut1 =:= FlowIn2,
           State =:= 4,
           Condition =:= bad.

      mode:- % Valve leaking
```

```
         FlowOut1 =\= FlowIn2,
         State =:= 6,
         Condition =:= bad.
}

schema solenoid:component
{

% type declarations

     pressRange Press1.  flowRateRange FlowOut1. conditionType FlowCond1.
     pressRange Press2.  flowRateRange FlowIn2.  conditionType FlowCond2.
     voltageRange Volt3. currentRange CurrentIn3.
     voltageRange Volt4. currentRange CurrentOut4.

% accessors for persistant variables

      terminal1(Press1,FlowOut1,FlowCond1).
      terminal2(Press2,FlowIn2,FlowCond2).
      terminal3(Volt3,CurrentIn3).
      terminal4(Volt4,CurrentOut4).

      press1(Press1). flowOut1(FlowOut1). flowCond1(FlowCond1).
      press2(Press2). flowIn2(FlowIn2).   flowCond2(FlowCond2).
      volt3(Volt3).   currentIn3(CurrentIn3).
      volt4(Volt4).   currentOut4(CurrentOut4).

% modes of operation

    order mode.

    mode:- % good state (closed solenoid)
         FlowOut1 < 5,
         FlowIn2 < 5,
         Volt3 - Volt4 >= 9,    % duty applied from ECM
         Volt3 - Volt4 <= 16,
         CurrentIn3 =:= CurrentOut4,
         State =:= 0,
         Condition =:= good.

    mode:- % good state (open solenoid)
         FlowOut1 =:= FlowIn2,
         FlowCond1 =:= FlowCond2,
         Press1 =:= Press2,
         CurrentIn3 < 5,     % Duty is not applied from ECM
         CurrentOut4 < 5,
         Volt3 =:= Volt4,
         State =:= 1,
         Condition =:= good.

    mode:- % blocked state
         FlowOut1 < 5,
         FlowIn2 < 5,
         State =:= 2,
         Condition =:= bad.

    mode:- % stuck open state
         FlowOut1 =:= FlowIn2,
        FlowCond1 =:= FlowCond2,
        Press1 =:= Press2,
        State =:= 3,
        Condition =:= bad.

    mode:- % restricted state
        FlowOut1 =:= FlowIn2,
        FlowCond1 =:= FlowCond2,
        Press1 =\= Press2,
        State =:= 4,
        Condition =:= bad.

    mode:- % leaking state
        FlowIn2 =\= FlowOut1,
        State =:= 5,
        Condition =:= bad.

    mode:- % contaminating state
        FlowCond1 =\= FlowCond2,
        State =:= 6,
        Condition =:= bad.
}


schema ccp:composite
{

% type declarations

    canister Can.  pcv PCV.     solenoid Sol.
    hose H1.      hose H2.      hose H3.
    hose H5.      hose H6.
    wire W1.      wire W2.

    pressRange Press1. flowRateRange FlowOut1. conditionType FlowCond1.
    voltageRange Volt2.   currentRange CurrentIn2.
    voltageRange Volt3.   currentRange CurrentOut3.
    pressRange Press4. flowRateRange FlowIn4.  conditionType FlowCond4.
    pressRange Press5. flowRateRange FlowIn5.  conditionType FlowCond5.
    pressRange Press6. flowRateRange FlowOut6. conditionType FlowCond6.

% accessors for persistant variables

     terminal1(Press1,FlowOut1,FlowCond1).
     terminal2(Volt2,CurrentIn2).
     terminal3(Volt3,CurrentOut3).
     terminal4(Press4,FlowIn4,FlowCond4).
     terminal5(Press5,FlowIn5,FlowCond5).
     terminal6(Press6,FlowOut6,FlowCond6).

     press1(Press1). flowOut1(FlowOut1). flowCond1(FlowCond1).
     volt2(Volt2).   currentIn2(CurrentIn2).
     volt3(Volt3).   currentOut3(CurrentOut3).
     press4(Press4). flowIn4(FlowIn4).   flowCond4(FlowCond4).
     press5(Press5). flowIn5(FlowIn5).   flowCond5(FlowCond5).
     press6(Press6). flowOut6(FlowOut6). flowCond6(FlowCond6).

    can(Can).    pCV(PCV).     sol(Sol).
    h1(H1).     h2(H2).     h3(H3).
    h5(H5).     h6(H6).

    order mode.
    mode:-
        Can:condition(good),
        PCV:condition(good),
        Sol:condition(good),
        H1:condition(good),
        H2:condition(good),
        H3:condition(good),
        H5:condition(good),
        H6:condition(good),
        W1:condition(good),
        W2:condition(good),
        condition(good).

    mode:- condition(bad).

% define components of system

    build:- Can isa canister,     Can:name(canister),
        PCV isa pcv,          PCV:name(presCV),
        Sol isa solenoid,      Sol:name(solenoid),
        H1 isa hose,          H1:name(hose1),
        H2 isa hose,          H2:name(hose2),
        H3 isa hose,          H3:name(hose3),
        H5 isa hose,          H5:name(hose5),
        H6 isa hose,          H6:name(hose6),
        W1 isa wire,           W1:name(wire1),
        W2 isa wire,           W2:name(wire2),

        Clist = [Sol,Can,PCV,H1,H2,H3,H5,H6,W1,W2],

% input/ouput parameters of system

          H1:terminal2(Press1,FlowOut1,FlowCond1),

          W1:terminal1(Volt2,CurrentIn2),

          W2:terminal2(Volt3,CurrentOut3),

          Can:terminal2(Press4,FlowIn4,FlowCond4),

          H5:terminal1(Press5,FlowIn5,FlowCond5),

          H6:terminal2(Press6,FlowOut6,FlowCond6),

% connections between components within system

          Sol:terminal1(PressA,FlowA,CondA),
          H1:terminal1(PressA,FlowA,CondA),

         Sol:terminal2(PressB,FlowB,CondB),
         H2:terminal2(PressB,FlowB,CondB),

          Sol:terminal3(VoltC,CurrentC),
          W1:terminal2(VoltC,CurrentC),
        Sol:terminal4(VoltD,CurrentD),
        W2:terminal1(VoltD,CurrentD),

        Can:terminal1(PressE,FlowE,CondE),
        H2:terminal1(PressE,FlowE,CondE),

        Can:terminal3(PressF,FlowF,CondF),
        H3:terminal2(PressF,FlowF,CondF),

        PCV:terminal1(PressG,FlowG,CondG),
        H3:terminal1(PressG,FlowG,CondG),

        PCV:terminal2(PressH,FlowH,CondH),
        H5:terminal2(PressH,FlowH,CondH),

        PCV:terminal3(PressI,flowRateRange FlowI,CondI),
        H6:terminal1(PressI,flowRateRange FlowIR,CondI),
       FlowIR < 5.0 - FlowI,

       composite::build.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                    Abductive Part:                %%

   m0(P1,F1):-
       apply_data0(P1,F1),
      diagnoses.

   default:- print(accessing_deductive_model_ccp),
      Can:mode,
      PCV :mode,
      Sol :mode,
      H1:mode,
      H2:mode,
      H3:mode,
      H5:mode,
      H6:mode,
      W1:mode,
      W2:mode.

   apply_data0(P1,F1):-
       press1(P1),
       flowOut1(F1).

   diagnoses:-
        solution_list(Clist,[],PrintList),
       print(PrintList).

    solution_list([],PrintList,PrintList).
   solution_list([component Hcomp|Tcomps],PrintList,List1):-
       Hcomp:condition(Ccond),
       Hcomp:state(Cstate),
      Hcomp:name(Cname),
       solution_list(Tcomps,[[Cname,Cstate]|PrintList],List1).

}
```

```
schema battery:component
{
    voltageRange Volt. currentRange CurrentOut.

% accessors for persistant variables

     terminal(Volt,CurrentOut).
     volt(Volt). currentOut(CurrentOut).

% modes of operation

    mode:- % good state, voltage between 9 and 16 volts
        Volt >= 9,
        Volt <= 16,
        State =:= 0,
        Condition =:= good.

    mode:- % bad state, undercharged
        Volt < 9,
        State =:= 1,
        Condition =:= bad.

    mode:- % bad state, overcharged
        Volt > 16,
        State =:= 2,
        Condition =:= bad.
}


schema igswitch:component
{
    switch Switch. junction3 Junction.
    voltageRange Volt1. currentRange CurrentIn1.
    voltageRange Volt2. currentRange CurrentOut2.
    voltageRange Volt3. currentRange CurrentOut3.
    signalType Signal.

% accessors for persistant variables

     terminal1(Volt1,CurrentIn1).
     terminal2(Volt2,CurrentOut2).
     terminal3(Volt3,CurrentOut3).
     volt1(Volt1). currentIn1(CurrentIn1).
     volt2(Volt2). currentOut2(CurrentOut2).
     volt3(Volt3). currentOut3(CurrentOut3).
    signal(Signal).

% define components of system

    mode:-  Switch isa switch,
        Junction isa junction3,

% input/output parameters of system

         Switch:terminal1(Volt1, CurrentIn1),
        Switch:signal(Signal),

         Junction:terminal2(Volt2,currentRange I2), I2 < 5.0-CurrentOut2,
% connections between components within system

            Switch:terminal2(VoltA,CurrentA),

            Junction:terminal1(VoltA,CurrentA),

% choose mode of operation for each component

          Switch:mode,

          Junction:mode,

          findmode.

% determine Condition and State of ignition switch

      findmode:-
            Switch:state(0),
            Junction:state(0),
            Signal =:= off,
            State =:= 0,
            Condition =:= good.

      findmode:-
            Switch:state(1),
            Junction:state(0),
            Signal =:= on,
            State =:= 1,
            Condition =:= good.

      findmode:-
            Switch:state(2),
            Junction:state(1),
            State =:= 2,
            Condition =:= bad.
      findmode:-
            Switch:state(2),
            Junction:state(2),
            State =:= 3,
            Condition =:= bad.
      findmode:-
            Switch:state(2),
            Junction:state(3),
            State =:= 4,
            Condition =:= bad.
      findmode:-
            Switch:state(2),
            Junction:state(4),
            State =:= 5,
            Condition =:= bad.
      findmode:-
            Switch:state(3),
            Junction:state(1),
            State =:= 6,
            Condition =:= bad.
      findmode:-
            Switch:state(3),
            Junction:state(2),
            State =:= 7,
            Condition =:= bad.
      findmode:-
            Switch:state(3),
            Junction:state(3),
            State =:= 8,
            Condition =:= bad.
      findmode:-
            Switch:state(3),
            Junction:state(4),
            State =:= 9,
            Condition =:= bad.
}


schema power:composite
{
% type declarations :

      battery Battery.  igswitch IS.  fuse ECM_F.
      wire W1. wire W2. wire W3.
      wire W4. wire W5. wire W6. wire W7.
      junction3 J1. junction3 J2.

      voltageRange Volt1.   currentRange CurrentOut1.
      voltageRange Volt2.   currentRange CurrentOut2.
      voltageRange Volt3.   currentRange CurrentOut3.
      voltageRange Volt4.   currentRange CurrentOut4.

      signalType Signal.
      real  U1.  real   U2.  real   V1.  real   V2.
      real  V3.  real   M3.  real   M4.  real   C1.

% accessors for persistant variables

       terminal1(Volt1,CurrentOut1).
       terminal2(Volt2,CurrentOut2).
       terminal3(Volt3,CurrentOut3).
       terminal4(Volt4,CurrentOut4).
       volt1(Volt1).   currentOut1(CurrentOut1).
       volt2(Volt2).   currentOut2(CurrentOut2).
       volt3(Volt3).   currentOut3(CurrentOut3).
       volt4(Volt4).   currentOut4(CurrentOut4).

      igswitch(IS).  batt(Battery).  ecm_f(ECM_F).
      w1(W1).       w2(W2).       w3(W3).
      w4(W4).       w5(W5).       w6(W6). w7(W7).
      j1(J1).      j2(J2).
      signal(Signal).

      order mode.
       mode:- print(accesing_default_of_Power_System),
        IS:condition(good),
        Battery:condition(good),
        W1:condition(good),
        W2:condition(good),
        W3:condition(good),
        W4:condition(good),
        W5:condition(good),
        W5:condition(good),
        W7:condition(good),
        ECM_F:condition(good),
        J1:condition(good),
        J2:condition(good),
        condition(good).

       mode:- print(accesing_default_of_Power_System),
        condition(bad).

   % define components of system

       build:- IS isa igswitch,      IS:name(ignitionswitch),
            Battery isa battery,   Battery:name(battery),
            W1 isa wire,          W1:name(wire1),
            W2 isa wire,          W2:name(wire2),
            W3 isa wire,          W3:name(wire3),
            W4 isa wire,          W4:name(wire4),
            W5 isa wire,          W5:name(wire5),
            W6 isa wire,          W6:name(wire6),
            W7 isa wire,          W7:name(wire7),
            J1 isa junction3,      J1:name(junction1),
            J2 isa junction3,      J2:name(junction2),
            ECM_F isa fuse,        ECM_F:name(ecmfuse),

     Clist = [IS,Battery,W1,W2,W3,W4,W5,W6,W7,J1,J2,ECM_F],

   % input/output parameters of system

            W7:terminal2(Volt2,currentRange I2), 0-CurrentOut2 =:= I2,
            J2:terminal3(Volt3,currentRange I3), 0-CurrentOut3 =:= I3,

            W3:terminal2(Volt1,currentRange I1), 0-CurrentOut1 =:= I1,
            W4:terminal2(Volt4,currentRange I4), 0-CurrentOut4 =:= I4,

            IS:signal(Signal),

   % connections between components within system

             Battery:terminal(VoltA,CurrentA),
             W1:terminal1(VoltA,CurrentA),

            W1:terminal2(VoltB,CurrentB),
            J1:terminal1(VoltB,CurrentB),

            J1:terminal2(VoltC,currentRange CurrentC),
             W2:terminal1(VoltC,currentRange CurrentCC),
            0-CurrentC =:= CurrentCC,

            J1:terminal3(VoltD,currentRange CurrentD),
            W5:terminal1(VoltD,currentRange CurrentDD),
            0-CurrentDD =:= CurrentDD,

            W5:terminal2(VoltE,CurrentE),
            IS:terminal1(VoltE,CurrentE),

            IS:terminal2(VoltF,CurrentF),
        W6:terminal1(VoltF,CurrentF),

        W2:terminal2(VoltH,CurrentH),
        ECM_F:terminal1(VoltH,CurrentH),

        ECM_F:terminal2(VoltI,CurrentI),
        W3:terminal1(VoltI,CurrentI),
        W4:terminal1(VoltI,CurrentI),

        W6:terminal2(VoltJ,currentRange CurrentJ),
        J2:terminal1(VoltJ,currentRange CurrentJJ),
       0-CurrentJJ =:= CurrentJJ,

        J2:terminal2(VoltK,currentRange CurrentK),
        W7:terminal1(VoltK,currentRange CurrentKK),
       0-CurrentKK =:= CurrentKK,

       composite::build.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                    Abductive Part:                    %%

    owner_complaint(diesel ling):-
            ECM_F:condition(bad).

    owner_complaint(hard_start):-
            IS:condition(bad).

    observation_1(Signal,V1,V2,V3):-
            apply_data0(Signal,V1,V2,V3),
            generate_diagnoses.

    observation_1(Signal,V1,V2,V3,M4):-
            apply_data0(Signal,V1,V2,V3,M4),
            generate_diagnoses.

    observation_1(Signal,V1,V2,V3,M3,M4):-
            default,
             apply_data0(Signal,V1,V2,V3,M3,M4),
            generate_diagnoses.

    observation_2(M4):- apply_data2(M4),
              generate_diagnoses.

    default:- print(accessing_deductive_model_power),
        IS:mode,
         Battery:mode,
        W1:mode,
        W2:mode,
        W3:mode,
        W4:mode,
        W5:mode,
        W6:mode,
        W7:mode,
         ECM_F:mode,
        J1:mode,
        J2:mode.
```

```
apply_data0(Signal,V1,V2,V3):-
        signal(Signal),
        volt1(V1),
        volt2(V2),
        volt3(V3).

apply_data0(Signal,V1,V2,V3,M4):-
        signal(Signal),
        volt1(V1),
        volt2(V2),
        volt3(V3),
         J2:terminal2(M4,MM4).

apply_data0(Signal,V1,V2,V3,M3,M4):-
        signal(Signal),
        volt1(V1),
        volt2(V2),
        volt3(V3),
         ECM_F:terminal2(M3,MM3),
         J2:terminal2(M4,MM4),
         apply_rule0(Signal,V1,V2,V3,M3,M4).
        a_priori.

apply_data2(M4):-
     J2:terminal2(M4,MM4),
     apply_rule2(M4).

 apply_rule0(Signal,V1,V2,V3,M3,M4):- print(rule1),
         Signal =:= on,
        V2 > 10,
        V1 < 5,
        M3 > 10,
         ECM_F:condition(bad).

 apply_rule0(Signal,V1,V2,V3,M3,M4):- print(rule1),
         Signal =:= on,
        V1 < 5,
        V2 < 5,
        V3 > 10,
         W7:condition(bad).

 apply_rule0(Signal,V1,V2,V3,M3,M4):- print(rule1),
         Signal =:= on,
        V1 < 5,
        V2 < 5,
        V3 > 10,
         W1:state(2).

 apply_rule0(Signal,V1,V2,V3,M3,M4):- print(rule1),
         Signal =:= on,
        V1 > 10,
        V2 < 5,
         J2:condition(bad).

apply_rule2(M4):-
        M4 > 10,
         J1:condition(good).
```

```
    a_priori:- W7:condition(bad).
    a_priori:- Battery:condition(bad).
    a_priori:- IS:condition(bad).
    a_priori:- J2:condition(bad).

    measure1(C1,C2):-
    print(rule0_4),
          apply_data1(C1,C2),
          apply_rule1.

    apply_data1(C1,C2):-
         currentOut1(C1),
         currentOut5(C2).

    apply_rule1:-
        C1 > 10,
         J1:condition(good).

    apply_rule1:-
        C1 > 0,
         J1:condition(bad).

    apply_rule1:-
    print(apply_apriori),
        a_priori_2.

    a_priori_2:- Battery:state(1).
    a_priori_2:- IS:condition(bad).
    a_priori_2:- ECM_F:condition(bad).

    generate_diagnoses:-
          solution_list(Clist,[],PrintList),
         print(PrintList).

      solution_list([],PrintList,PrintList).
     solution_list([component Hcomp|Tcomps],PrintList,List1):-
         Hcomp:condition(Ccond),
         Hcomp:state(Cstate),
         Hcomp:name(Cname),
          solution_list(Tcomps,[[Cname,Cstate]|PrintList],List1).

}

schema vehicle:composite
{
% type declarations :
   ccp CCP.
   power Power.

    voltageRange Volt1.   currentRange Current1.
    voltageRange Volt2.   currentRange Current2.

   order mode.
   mode:- Power:condition(good),
        CCP:condition(good),
        condition(good).

   mode:- condition(bad).
```

```
% system components :
     build:- Power isa power, Power:name(power_system),
          CCP isa ccp,    CCP:name(ccp_system),

          Clist = [Power,CCP],

% connections between components within system :

            Power:terminal2(VoltA,CurrentA),
            CCP:terminal2(VoltA,CurrentA),
          %default,

          composite::build.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%                          Abductive Part :                          %%

% containing rule at level of Vehicle

      owner_complaint(U1):- Power:owner_complaint(U1).

     observation_1(Signal,V1,V2,V3,C1,C2):-
          apply_data0(Signal,V1,V2,V3,C1,C2),
          generate_diagnoses.

     observation_1(Signal,V1,V2,V3,M4,C1,C2):-
          apply_data0(Signal,V1,V2,V3,M4,C1,C2),
          generate_diagnoses.

     observation_1(Signal,V1,V2,V3,M3,M4,C1,C2):-
          default,
          apply_data0(Signal,V1,V2,V3,M3,M4,C1,C2),
          generate_diagnoses.

      observation_2(M4):- Power:observation_2(M4).

     default:-
          Power:mode,
          CCP:mode.

     apply_data0(Signal,V1,V2,V3,C1,C2):-
          Power:observation_1(Signal,V1,V2,V3),
          CCP:m0(C1,C2).

     apply_data0(Signal,V1,V2,V3,M4,C1,C2):-
          Power:observation_1(Signal,V1,V2,V3,M4),
          CCP:m0(C1,C2).

     apply_data0(Signal,V1,V2,V3,M3,M4,C1,C2):-
          Power:observation_1(Signal,V1,V2,V3,M3,M4),
          CCP:m0(C1,C2).

     generate_diagnoses:-
           solution_list(Clist,[],PrintList),
          print(PrintList).

       solution_list([],PrintList,PrintList).
      solution_list([component Hcomp|Tcomps],PrintList,List1):-
```

```
        Hcomp:condition(Ccond),
        Hcomp:state(Cstate),
        Hcomp:name(Cname),
         solution_list(Tcomps,[[Cname,Ccond]|PrintList],List1).

}
vehicle Vehi.
```

Data File for Run Test:

```
load sys_vehicle.kb
load c1.kb

Vehi isa vehicle.
Vehi:build.

Vehi:owner_complaint(dieselling).
Vehi:owner_complaint(hard_start).
Vehi:observation_1(off,0.0,0.0,11.9,-1,0).
Vehi:observation_1(on,0.0,0.0,11.9,-1,0).
Vehi:observation_1(on,0.0,0.0,11.9,11.9,-1,0).
Vehi:observation_1(on,0.0,0.0,11.9,0.0,11.9,-1.0,0.0).
Vehi:observation_2(11.9).
```

References

[Abu-Hanna,88] Abu-Hanna, A. and Gold, Y., An Integrated, Deep-shallow expert system for multi-level diagnosis of dynamic systems, in: J.S. Gero (ed.), Artificial Intelligence in Engineering: Diagnosis and Learning, Southhampton,75-94, 1988.

[Aho,74] Aho, A.V., J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley,1974.

[Bratko,86] Bratko, I. Prolog: Programming for Artificial Intelligence, Addison-Wesley, 1986.

[Buchanan,84] Buchanan, Shortliffe (eds), Rule-Based Expert Systems : The MYCIN Experiments of The Stanford Heuristics Programming Project, Addison-Wesley, 84.

[Chevrolet,89] 1989 Celebrity Service Manual, Chevrolet-GM

[Cohen,90] Cohen, J., Constraint Logic Programming, Communications of the ACM, 33(7), pp. 52-68, 90.

[Colmerauer,90] Colmerauer, A., An Introduction to Prolog III, Communications of the ACM, 33(7), pp.69-90, 90

[Dague, 82] Dague, P., Deves, P., Raiman, O., Troubleshooting: when Modelling is the Trouble, Proceedings of AAAI-87, 600-610.

[Davis,82] Davis, R., Shrobe, H., Hamscher, W., Wiecker, K., Shirley, M., and Polit, S., Diagnosis based on Structure and Function. In Proceedings of AAAI-82, Pittsburgh, Pennsylvania, 137-142.

[Davis,84] Davis, R., Diagnostic Reasoning Based on Structure and Behaviour, Artificial Intelligence 24 (1984) 347-410.

[Davis,88]	Davis, R., and Hamscher, W., Model-based Reasoning: Troubleshooting, in Exploring Artificial Intelligence, edited by H.E. Shrobe and the American Association for Artificial Intelligence, (Morgan Kaufman, 1988), 297-346.
[de Kleer,84]	de Kleer, J., An Assumption-Based Truth Maintenance System, Artificial Intelligence 28 (1986), 127-162.
[de Kleer,86]	de Kleer, J., Extending the ATMS, Artificial Intelligence 28 (1986), 163-196.
[de Kleer,87]	de Kleer, J. and Williams, B.C, Diagnosing Multiple Faults, Artificial Intelligence 32(1) (1987), 97-130.
[de Kleer,89a]	de Kleer, J. and Williams, B.C, Diagnosis with Behavioral Modes, in: Proceedings IJCAI-89, Detroit, MI (1989), 1324-1330.
[de Kleer,89b]	de Kleer, J., A Comparison of ATMS and CSP Techniques, Proceedings IJCAI-89, Detroit, MI (1989), 290-296.
[de Kleer,90a]	de Kleer, J., Exploiting Locality in a TMS, In Proceedings of AAAI-90, 1990.
[de Kleer,90b]	de Kleer, J., A.K. Mackworth, R. Reiter, Characterizing Diagnoses, In Proceedings of AAAI-90, 1990.
[de Kleer,90c]	de Kleer, J., Using Crude Probability Estimates to Guide Diagnosis, Research Note, Artificial Intelligence, 45(3), 1990.
[Doyle,79]	Doyle, J., A Truth Maintenance System, Artificial Intelligence 12 (1979), 231-272.
[Duffy,88]	Duffy, J.E., Auto Engines, The Goodheart-Willcox Company, Inc, 1988.
[Finin,89]	Finin, T.; Morris, G., Abductive Reasoning in Multiple Fault Diagnosis, Artificial Intelligence Review, 1989(3), 129-158.
[Fink,86]	Fink, P.K.; Lusth, J.C., A Second Generation Expert System for Diagnosis and Repair of Mechanical and Electrical Devices, SAE Technical Paper Series, 860337, 86.
[Forbus,88]	Forbus, K.D., Qualitative Physics: Past, Present, and Future, in Exploring Artificial Intelligence, edited by H.E.

Shrobe and the American Association for Artificial Intelligence, (Morgan Kaufman, 1988), 239-296.

[Forbus,88] Forbus, K.D., Intelligent Computer-Aided Engineering, AI Magazine, Fall- 88.

[Genesereth,82] Genesereth, M.R., Diagnosis Using Hierarchical Design Models, in Proceedings AAAI-82, 278-283.

[Genesereth,84] Genesereth, M.R., The Use of Design Descriptions in Automated Diagnosis, Artificial Intelligence 24 (1984), 411-436.

[Hamscher,84] Hamscher, W. and Davis, R., Diagnosis Circuit with state: An inherently Underconstrained Problem, Proceedings of AAAI-84, 142-147.

[Hamscher,87] Hamscher, W. and Davis, R., Issues in Model-Based Troubleshooting. Memo 893, MIT Artificial Intelligence Laboratory, 1987.

[Hamscher,89] Hamscher, W., Temporally Coarse Representation of Behavior for Model-based Troubleshooting of Digital Circuit, Proceedings IJCAI-89, Detroit, MI (1989).

[Hamscher,90a] Hamscher, W., XDE: Diagnosing Devices with Hierarchic Structure and Known Component Failure Modes. IEEE Conference on AI Applications, 1990.

[Hamscher,90b] Hamscher, W., Modelling Digital Circuits for Troubleshooting: An Overview, IEEE Conference on AI Application, March 90

[Havens,83] Havens, W.S., Recognition Mechanisms for Schema-based Knowledge Representations, Comp. & Math. with Appls., vol 9, No 1, pp 185-189, 1983.

[Havens,90a] Havens, W.S., J. Jones, C. Hunter, S. Joseph, A. Manaf, Model-Based Automotive Diagnosis using the Echidna Constraint Reasoning System, Pergamon Press Article, 1990.

[Havens,90b] Havens, W.S., S. Sidebottom, G. Sidebottom, J. Jones, M. Cuperman, R. Davison, Echidna Constraint Reasoning System: Next Generation Expert System Technology, Simon Fraser University Technical Report, CSS-IS TR 90-09.

[Havens,91] Havens, W.S., Dataflow Dependency Backtracking in a new CLP Language, proc AAAI Spring Symposium on Constraint Reasoning, Stanford, Ca, pp 110-127, March,1991.

[Jaffar,87a] Jaffar, J. and J.L. Lassez, Constraint Logic Programming, In Proc. Fourteenth ACM POPL Conf., Munich, 1987.

[Jaffar,87b] Jaffar, J. and S. Michaylov, Methodology and Implementation of a CLP System, In Proc. Fourth International Conference in Logic Programming, Melbourne, 1987.

[Joseph,89] Joseph, S., J. McCarney, A New Engine Analysis System for Sensor and Actuator Related Problems, Proc. Future Transportation Technology Conference and Exposition, SAE 891726, Vancouver, B.C., 1989.

[Kowalski,79] Kowalski, R., Logic for Problem Solving, The Computer Science Library, 1979.

[Klausmeier,86] Klausmeier, R., Using Artificial Intelligence in Vehicle Diagnosis Systems, SAE Technical Paper Series, 861124, 86.

[Kline,89] Kline, P.J., S.B. Dollins, Designing Expert Systems: A Guide to Selecting Implementation Techniques, John Wiley & Sons, 1989.

[Kuipers,84] Kuipers, B., Commonsense Reasoning about Causality: Deriving Behavior from Structure, Artificial Intelligence, 24 (1984), pp. 169-203.

[Lee, 90] Lee, M.H.; Hunt, J.E.; Price, C.J.; Long, F.W., REPAIR : A Model-Based Diagnosis System, UK IT 1990 Conference pp. 266-270, Southhamptom, UK, 1990.

[Mackworth,77] Mackworth, A.K., Consistency in Network Relations, Artificial Intelligence, 8, pp. 99-118, 1977.

[Mackworth,85] Mackworth, A.K., E.C. Freuder, The Complexity of Some Polynomial Network Consistency Algorithm for Constraint Satisfaction Problems, Artificial Intelligence, 25, pp.65-74, 1985.

[Meyer,88] Meyer, B., Object-oriented Software Construction, Prentice Hall, 1988.

[Nadel,90] Nadel, B.A., The Complexity of Constraint Satisfaction in Prolog,In Proceedings of AAAI-90, 1990.

[Pereira,87] Pereira, F.C.N.,and S.M. Shieber, Prolog in Natural-Language Analysis, CSLI lecture notes, 1987.

[Poole,88] Poole, D., A Logical Framework for Default Reasoning, Artificial Intelligence, 36 (1988), pp. 27-47.

[Provan,88] Provan, G.M., The Computational Complexity of Multiple-Context Truth Maintenance Systems, Technical Report 88-11, University of British Columbia, Department of Computer Science, 1988.

[Reiter,80] Reiter, R., A Logic for Default Reasoning,Artificial, Intelligence 13 (1980), 81-132.

[Reiter,87] Reiter, R., A Theory of diagnosis from the first principles, Artificial Intelligence 32 (1987), 57-95.

[Roth,67] Roth, J.P., W.G. Bouricius, P.R. Schneider, Programmed Algorithm to Compute Tests to Detect and Distinguish Faults in Logic Circuits, IEEE Transactions on Electronics Computers, Vol EC-16 No 5, October 1967.

[Scarls,85] Scarls, E., Jamieson, J.R. and Delaune, C.I., A Fault Detection and Isolation Method Applied to Liquid Oxygen Loading for Space Shuttle. In Proceedings IJCAI-85, Los Angeles, CA, 414-416.

[Shiga,88] Shiga, H., S. Mizutani, Car Electronics, Nippondenso Co, Ltd, 1988

[Sidebottom,91a] Sidebottom, G., W.S. Havens, Hierarchical Arc Consistency Applied to Numeric Constraints Processing in Logic Programming, Simon Fraser University Technical Report, in preparation.

[Sidebottom,91b] Sidebottom, S., W.S. Havens, M. Cuperman, R. Davison, G. Sidebottom, Echidna Constraint Reasoning System: Programming Guide, Tutorial and Manual Version 1, Simon Farser University, 1991.

[Struss,88] Struss, P., Extensions to ATMS-based Diagnosis, in: J.S. Gero (ed.), Artificial Intelligence in Engineering: Diagnosis and Learning, Southhampton, 3-28, 1988.

[Struss,89] Struss, P. and Dressler, O., "Physical Negation"- Integrating Fault Fodels into the General Diagnosis Engine, in: Proceedings IJCAI-89, Detroit, MI (1989), 1318-1323.

[Tomikashi,87] Tomikashi, M.; Kishi, N.; Kanegae, H.; Hino, A., Application of an Expert System to Engine Troubleshooting, SAE Technical Paper Series, 870910, 1987.

[Wu,90] Wu, T.D., Efficient Diagnosis of Multiple Disorders Based on a Symptom Clustering Approach, In Proceedings of AAAI-90, 1990.

[William,90] William, B.C., Interaction-based Invention: Designing Novel Devices from First Principles, In Proceedings of AAAI-90, 1990.