VMES: A Versatile Maintenance Expert System[1]

J. Chen J. Choi J. Geller A. Kumar M. R. Taie
S. C. Shapiro S. N. Srihari S. J. Upadhyaya

Department of Computer Science
State University of New York at Buffalo
Buffalo, New York 14260

November 20, 1989

[1]This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, and the Air Force Office of Scientific Research, Bolling AFB DC 20332 under Contract No. F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium (NAIC).

Abstract

This paper describes a versatile maintenance expert system (VMES) that has been designed with a focus on applied knowledge representation. Two main points of interest are described, the representation and reasoning mechanisms necessary for diagnosis based on a deep model of a device, and the representation for an integrated graphical user interface with limited natural language capabilities. Device structure is represented in a hierarchy of device types. Structural templates and instantiation rules permit focused diagnostic reasoning using lazy instantiation. Functional description is procedurally attached to the declarative network representation. Similarly, pieces of graphical code are attached to a declarative representation of the graphical appearance of the device.

VMES is a device-model-based versatile maintenance expert system which assists the technician in isolating specific faulty components or connections in a malfunctioning digital circuit. A device representation formalism that supports the diagnostic reasoning of VMES and eases its adaptation to new devices is described. The salient feature of the scheme is the inclusion of both logical and physical structural descriptions of the target device. The two representations enable VMES to make efficient diagnostic judgements and to interact effectively with the user in performing repair and test. The user interface of VMES is treated as a separate area of scientific investigation. We describe the design and implementation of three interfaces, *viz.*, a graphics display interface, a graphics input interface, and a natural language input interface.

Index terms: Model-based troubleshooting, Versatile maintenance, Intelligent user interfaces.

Contents

1 Introduction 2

2 Knowledge of Versatile Diagnosis 3
2.1 Knowledge Analysis and Characterization 3
2.1.1 Device-Specific Empirical Associations 3
2.1.2 Generic Domain Knowledge 4
2.1.3 Knowledge Representing a Model of the Device 4
2.2 Core Knowledge of Versatile Fault Diagnosis 5
2.3 Logical and Physical Knowledge in Diagnosis 6

3 Device Representation 7
3.1 Structural Representation 7
3.1.1 Instantiation Rules as the Level-1 Abstraction 7
3.1.2 Structural Templates as the Level-2 Abstraction 10
3.1.3 Cross-Links between Logical and Physical Structures 13
3.2 Functional Representation 13
3.3 Graphics Knowledge Representation 15

4 Diagnostic Reasoning 15
4.1 Control Structure 15
4.2 Diagnosing Wires and POCONs 19
4.3 Repair Suggestion 20

5 Intelligent User Interface 20
5.1 The TINA Graphics Interface 20
5.2 The Readform Interface for Object Creation 22
5.3 The Natural Language Interface 22

6 Conclusion 23

A Representation and Diagnosis of Sequential Circuits 26
A.1 Introduction 26
A.2 Representation of Sequential Circuits 26
A.3 Handling Feedback in Sequential Circuits 26
A.4 Candidate Generation based on Electrical Behavior 27
A.5 Diagnosis of Sequential Circuits 29
A.6 Assumptions and their Relaxation 30
A.7 Conclusion 31

B A Scheme for Shadowing General Knowledge by Its Instances 32
B.1 Introduction 32
B.2 Automatic Migration of General to Specific Knowledge 33
B.3 Shadowing General Knowledge by Its Instances 35
B.4 An Implementation : SNePS 37
B.5 An Application 41
B.6 Conclusion 43

1 Introduction

VMES is a device-model-based versatile maintenance expert system for the domain of digital circuits [Shapiro *et al.*, 1986]. The objective of VMES is to interact with a maintenance technician (the user) to identify the specific faulty components or connections of a malfunctioning circuit. The versatility of a maintenance system is defined as ability to adapt to different devices without extensive knowledge engineering, capability of diagnosing a wide range of common faults, capability of operating at different maintenance levels, and capability of interacting with users through various media.

To achieve the desired versatility, VMES follows the device-model-based approach [Davis, 1984; Davis and Hamscher, 1988; Genesereth, 1984; de Kleer and Williams, 1987]. The device-model-based approach, as opposed to the empirical-rule-based approach used by MYCIN [Shortliffe, 1976] for medical diagnosis and by CRIB [Hartley, 1984] for computer hardware fault diagnosis, is suggested to have advantages in knowledge acquisition, diagnosis capability, and system generalization [Davis and Shrobe, 1983; Davis, 1984; Genesereth, 1984]. Digital circuits are chosen as the target domain for several reasons. First of all, it is practically important due to their widespread use, high introduction rate, and relatively short market life cycles of devices in the domain, and also because of the general shortage of qualified maintenance experts. Second, the domain is complex enough in that it consists of various types of devices with different structures and functionalities. Thus it is a good domain to test system versatility. Versatile fault diagnosis of digital circuits involves fault isolation (or "troubleshooting") and repair [Coppola, 1984]. Fault isolation is the task of locating the faulty part(s) in a malfunctioning device. Repairing a device in the domain of electronic circuits usually means replacing the identified faulty parts or fixing a bad contact point.

The architecture of VMES is illustrated in Figure 1. VMES is implemented in SNePS [Shapiro, 1979], the Semantic Network Processing System, and consists of five modules: the knowledge base, the inference engine, the active database, the user interface, and the builder interface. The knowledge base is implemented as an expandable component library which contains component descriptions. The descriptions of a device are arranged hierarchically to provide different abstraction levels of the device. As a result, the inference engine is able to focus on a limited number of objects at any time. The inference engine has the generic diagnosis knowledge of the domain, and uses SNIP, the SNePS inference package, as its basis [McKay and Shapiro, 1981; Shapiro, 1979]. An active database is created and updated throughout each diagnostic session to keep the instantiated objects and their associated diagnostic states and values. Parts of a device are instantiated only when needed so that unnecessary details are not included in the active database. The user interface interfaces the maintenance technicians when carrying out a diagnostic session. The builder interface interfaces the engineers or senior technicians to update the knowledge base for new devices.

As knowledge engineering is to empirical-rule-based system, device modeling/representation is the key to the success of a device-model-based fault diagnostic system, since knowledge about the structure and function of a device is the major knowledge source of reasoning in such a system. Consequently, our efforts are focused on the development of a device representation formalism for versatile maintenance. All knowledge, whether it is structural, functional, or graphical, is maintained in a unified SNePS knowledge base, User interaction is an important issue of the VMES project in two aspects: VMES has to communicate with the maintenance technician for test and repair, and it has to provide an engineer or a senior technician facilities for adapting it to other devices by adding their descriptions to the component library.

Structural and functional descriptions, usually referred to as "design models" of a device, have been suggested as a solution to the difficulties of empirical-rule-based diagnosis systems in knowledge acquisition, diagnosis capability, and system generalization [Davis, 1984; Genesereth, 1984]. Such systems are referred to as "device-model-based", "design-model-based", or "specification-based". Most device-model-based fault diagnosis systems use a structural device description only from a logical perspective. Human experts usually perform diagnostic reasoning based on a logical model of the target device, and form a repair plan based on its physical model. Sometimes the schematic diagram of a device is quite different from its physical form, and even experts may have difficulty in matching a schematic to the real object. In this work, both logical and physical structures of a device are represented, and innovative ways of using the physical structural description for fault diagnosis are explored. We develop a system which incorporates both physical and logical representations in a similar way to human experts — it reasons on the logical representation, but uses the physical representation to determine when to terminate the diagnosis process, and which physical

parts should be replaced or fixed. The system is superior to a human expert in that, by incorporating logical and physical representations and cross-links between them, it can clearly direct the user to the right place on the real device for test and repair.

We also find that explicit representation of wires and POCONs (points of contact) is necessary for diagnosing faults of circuit connections [Taie and Srihari, 1987]. The traditional model of a wire as a uni-directional module is inappropriate, because it ignores its bi-directional nature, and it does not include POCONs. In this work, a wire is modeled as a bi-directional module to preserve its physical property, and its uni-directional design intention is retained by the connection mechanism. Components are connected either by forming a POCON from two different ports or by superimposing two ports, which are the same port abstracted at two different hierarchical levels, together. With this new model, a fault diagnosis system is able to locate interrupted wires and bad contact points.

2 Knowledge of Versatile Diagnosis

The performance of an expert system depends mostly on the contents and the forms of its knowledge, since it is the major resource of its reasoning [Buchanan and Shortliffe, 1984; Hayes-Roth *et al.*, 1983; Michie, 1980]. In this section, we analyze the knowledge for troubleshooting electronic circuits, and identify the core knowledge for versatile fault diagnosis. We discuss how domain experts coordinate different views of a same device in diagnosing and repairing it, and the difficulties they might have in coordinating the different views.

2.1 Knowledge Analysis and Characterization

In analyzing the knowledge used in circuit diagnosis, we suggest that the knowledge be categorized as device-specific empirical associations, generic domain knowledge, and a device model.

2.1.1 Device-Specific Empirical Associations

Device-specific empirical associations relate observed symptoms to possible fault hypotheses for a specific device. An example is a rule from the REACTOR, which is a system for diagnosing nuclear accidents [Nelson, 1982]. The rule says:

IF ((PCS pressure decreasing) (HPIS on))
THEN (PCS integrity challenged)

Another example is the rules from the CRIB system for diagnosing computer hardware [Hartley, 1984]. These rules have the form of:

IF symptoms S_1 to S_n have been observed
THEN assume the fault is in sub-unit U_i.

Such rules are highly device specific, and cannot apply to other devices in the same domain, since devices in the same domain may have different structures.

Device-specific empirical associations are the knowledge about "how a device fails", in other words, they record the failure modes of a device. Though it is possible to learn this kind of knowledge from other domain experts, most maintenance technicians acquire it through their own experience with the device. Therefore, the process to accumulate this kind of knowledge is time consuming, and the knowledge is "private", since it is usually not well organized and sometimes hard to be shared with others. This kind of knowledge does not care about "how the device works" or "why this symptom implies that the particular component may be faulty". It only concerns the relationship between an observed symptom and possible fault hypothesis. Using this kind of knowledge for diagnosis, no matter in the medical domain or hardware domain, has been proven to be very effective.

2.1.2 Generic Domain Knowledge

Generic domain knowledge is the general knowledge in the designated domain that domain experts use for fault diagnosis. A very primitive piece of generic knowledge for circuit fault diagnosis is that "when an output of a device deviates from the expected value regarding to the known inputs, it implies that the device is malfunctioning". Another one is that "to locate the faulty component(s), one should first identify the signal flow paths from inputs to the bad output, this can be efficiently done by back-tracing the connections from the bad output to the relevant inputs". The domain knowledge is not necessarily very primitive, it may also relate to higher human perception as in the example of "a burnt appearance of a component implies that the component is a potential faulty part".

Though sometimes the generic domain knowledge is still considered as a kind of empirical association, it is not device-specific and it is "public" knowledge in the domain. It can be applied to any device in the domain, and it can be easily shared among the experts in the domain. Novices in the domain can learn it from experienced technicians, or even from some books or manuals.

In general, the generic domain knowledge for circuit fault diagnosis is the general principles in searching for the faults through the structure of the target device. The examples mentioned above represent part of the general search strategies. Other well-known techniques include "when tracking down a missing or distorted signal, bisect the path into two groups with equal number of components or equal fault probabilities", "when selecting a test, take the cost of the test into account", and "first check the component which contributes to more bad outputs or which have an outstanding failure record".

2.1.3 Knowledge Representing a Model of the Device

A model of the target device, which consists of the structural and functional information about the device, is maintained by the technician when troubleshooting electronic circuits. Though the model is device specific, unlike device-specific empirical associations, a model of a device is "public" knowledge which is highly structured and readily available at the time when the device is designed. This model is sometimes referred to as a "mental model" of the device since it is the technician's view of a device when troubleshooting it [Rasmussen and Jensen, 1974]. This model is also referred to as a "design model" of the device, since it usually reflects, though not always necessarily, the design of the device [Genesereth, 1982].

Previous research shows that technicians use a hierarchical model of the target device for troubleshooting electronic equipment [Dale, 1957; Rasmussen and Jensen, 1974]. It has also been shown that engineers tend to design new devices in a hierarchical manner [Genesereth, 1982]. Though not all electronic devices have a hierarchical structure, in general, it is quite natural to view a device as a set of hierarchically arranged subunits. Hierarchical structure is the favorite structure of both maintenance technicians and design engineers not only due to its naturalness, but also because that a hierarchical structure facilitates the focus when they are carrying out their tasks, and thus the jobs can be done more efficiently. In other words, the task can be done more easily since the hierarchical structure allows a technician or an engineer to limit his attention to merely a small well-defined part of the whole device, which reduces the mental load involved in the task.

In a model of device, knowledge about the structure and the function of the device is maintained. Structural knowledge can be further divided into logical and physical structural knowledge. Logical structure is based on how the device carries out its designated function, i.e., it is based on the functionality of the device. Conventional representation media for logical structure include schematic diagrams and high level block diagrams. Parts or subunits on these diagrams are marked with a name which associates a particular function to it. Connections show the relevant signal flows with arrows to indicate the intended direction, though in reality, signals can flow on a wire in either direction. Ports where signals flow into or out of a component are clearly drawn. Irrelevant information is omitted (e.g., unused ports of a chip).

Physical structure is based on how the device is assembled. The conventional representation of physical structure is the assembly (manufacture) layouts or even just a picture of the device. Rather than the functionality of the components, a physical structure is interested in the real appearance of the device, especially in the relative physical relationship between components. As for logical structure, maintenance technicians tend to use a hierarchical representation for physical structure. Subunits at each level are abstracted in a way that they are corresponding to easily-recognized entities which can be replaced as a single unit in some maintenance levels. A simple, but maybe most widely used, prototype is to abstract the

physical structure of a device as a hierarchy of system, cabinets, modules, boards, and chips.

Besides the logical structure and the physical structure, another important knowledge in the model of a device is the functional knowledge of the device. Functional descriptions are associated with the objects (the device and its subunits) at every level of the hierarchical logical structure. In the electronic circuits domain, it is the knowledge about the relationship between the inputs and the outputs of an object. Conventional representation for functional knowledge is mathematical equations, truth or numerical tables, and pictorial representation as waveforms.

Another important part of a device model is the knowledge about how the logical structure, the physical structure and the function of the device are linked together. The functional description are closely associated with the logical structure, and this link can be easily realized via the component type names on the schematic diagrams. Since the logical structure and the physical structure may be quite different for some devices, to maintain the links between these two structures increases the mental load of technicians. In fact, we observe that maintenance technicians frequently have difficulty in locating a point on the schematic diagram (logical structure) on the real device (physical structure). Reasons are that to maintain the links between the logical structure and the physical structure will overload the mental capacity of maintenance technicians, and moreover, these links are not well-documented in most cases.

2.2 Core Knowledge of Versatile Fault Diagnosis

In analyzing the knowledge used by maintenance technicians for troubleshooting electronic equipment, it turns out that human experts use all kinds of knowledge described above, *viz.*, device-specific empirical associations, generic domain knowledge, and a model of the target device, in an intermixed manner.

When diagnosing a device one is familiar with, an experienced technician uses many device-specific empirical associations, which directly relate the observed symptoms (behaviors) to possible faulty components, to quickly pinpoint the fault by skipping lots of detailed casual links. But he also refers to structural information of the device to help him carry out the diagnosis. Other knowledge comes into play at points when device-specific empirical associations fall short, but the use of device-specific empirical associations may be resumed later on if it is proper.

On the other hand, when troubleshooting a device never seen before (which includes newly designed devices), the technician has virtually no device-specific knowledge about how the device fails. An experienced technician retains his competence by reasoning directly on a model of the device and by using the generic domain knowledge. Both kinds of knowledge are required for the diagnosis, and the generic domain knowledge guides the search on the structure of the device represented in the model. The model of the device also provides the technician with the functional knowledge of the device, which is needed by the technician to judge the results of a test.

Obviously, it is desirable to represent and use all sorts of knowledge efficiently in an automatic fault diagnosis system. Unfortunately, current artificial intelligence technique is still far behind this application, and no existing fault diagnosis system uses all of them: some are based on purely empirical associations associations [Hartley, 1984], and some are mainly based on a model of the device, i.e., the structural and functional descriptions of the device [Davis *et al.*, 1982; Genesereth, 1982].

In attempting to combine all knowledge to facilitate diagnosis, a two-level architecture has been proposed for neurological diagnosis [Xiang and Srihari, 1986], which suggests that the system that the system first works on the empirical-association-based module, and then turns to the device-model-based module if the problem can not be successfully solved by the first module. We do not adopt this idea since the focus of our research is to develop a versatile fault diagnosis system, and the inclusion of empirical associations at this point impairs the construction of such a versatile system.

As mentioned before, versatility of an automatic fault diagnosis system is extremely important in an electronic circuit domain due to the fast rate at which new products are introduced and their relatively short market lives. In noticing that an experienced technician is able to effectively troubleshoot an electronic device by using the schematic diagrams of the device and his general knowledge about troubleshooting devices in the domain and without having to learn how the device may fail, we define an automatic versatile fault diagnosis system as an expert system which behaves like an experienced technician who is competent in diagnosing devices he has never seen before. A major point here is that a versatile fault diagnosis system

should be able to adapt to new devices easily, just like an experienced technician should.

Device-specific empirical associations are quite different from the knowledge of a device model in both the contents and the representation forms. Device-specific empirical associations are assertive knowledge (or propositions) relating symptoms to possible faults. It is natural to represent them as production rules. Knowledge of a device model is basically descriptive, and can be best represented as semantic networks or as frames. There are two major hurdles in including the device-specific empirical associations in a versatile fault diagnosis system: techniques in acquiring it by interviewing with domain experts through knowledge engineering have to be improved so that this process will not slow down the adaptation of the system to other devices; and the capability of an expert system in selecting and using proper knowledge at proper time from a knowledge-base (or knowledge-bases) containing various types of knowledge in different forms has to be achieved so that the system can have an acceptable performance.

One major consideration in developing a versatile fault diagnosis system is the system's ability in adapting itself to other devices. It is improper to include the device-specific empirical associations in a versatile maintenance system, since this may impair system versatility, and moreover, this kind of knowledge is not available at all for newly designed devices. Therefore, only the generic domain knowledge and the knowledge of a device model should be incorporated into a versatile fault diagnosis system. In mimicking the versatility of experienced maintenance technicians in troubleshooting devices they had never seen before, the generic domain knowledge is transformed into the search algorithms and diagnosis rules of the fault diagnosis system, and the knowledge of a device model, which is the basis of the system's reasoning, becomes the core knowledge of the system.

2.3 Logical and Physical Knowledge in Diagnosis

The emphasis of most previous research on the device-model-based approach to fault diagnosis has been on using the logical structure of a target device. Such a representation emphasizes the functional interrelations of components but not the physical interrelationships, e.g., functionally unrelated components may be physically related (adjacent, in the same area, etc.). However, knowledge about physical device structure often plays an important role in fault diagnosis performed by human technicians. This research explores the representation and use of knowledge about both logical and physical structures of target devices in a versatile maintenance system. In particular, we examine the relationships (cross-links) between logical and physical structures.

The use of physical structure in a diagnostic problem in the medical domain, *viz.*, neurological diagnosis based on a model of neural pathways in the human spinal cord, was explored by Xiang and Srihari [Xiang and Srihari, 1985]. In their system, two functionally unrelated paths may be examined due to their physical proximity. In the domain of circuit diagnosis there is little in the literature on physical structure representation with the exception of references made by Davis [Davis and Shrobe, 1983; Davis, 1984]. He suggests including a physical description based on the notion that different paths of interaction or adjacency should be represented to diagnose different kinds of faults. A particular application of utilizing the physical structure description of the device is demonstrated as the diagnosis of bridge faults under the assumption that bridges can only occur between two adjacent pins of an IC (integrated circuit) chip.

Human diagnosticians for electronic devices seem to simultaneously maintain models of the logical and physical structures of the target device. They carry out most of the diagnostic reasoning over the logical structure of the device due to its functional association. While carrying out the reasoning, the logical structure is apparently mapped to the physical structure from time to time. Tests and measurements are first initialized using the logical structure, and then are realized and executed on the physical structure. Repair, which is usually done by replacing a physical unit or by fixing a physical connection, is planned and done on the physical structure. In other words, maintenance technicians use a model of physical structure of the target device, which is a hierarchically arranged set of replaceable physical components at various maintenance levels such as field-level and depot-level. By mapping the logical structure of the device to its physical equivalent, maintenance technicians are able to terminate the diagnostic process at the right moment and to form an adequate repair plan.

Given that the mapping between the logical structure of the device and its physical equivalent happens throughout the diagnostic process at all hierarchical levels, the speed in carrying out the mapping is critical

to the time needed to locate faults. This implies that objects on both the logical level structure and the physical structure of the device should be closely linked to each other so that the mapping is done efficiently. Even experienced technicians may have difficulty in locating a point of a schematic diagram on the real device, where the schematic diagram represents the logical structure of the device, and the form of the real device is the physical structure. This is attributable to a lack of cross-links at all hierarchical levels of the device in human memory. On the other hand, when modeling and representing a device in an automatic fault diagnosis system, the cross-links between its logical structure and physical structure can be modeled and represented to an appropriate level of detail.

3 Device Representation

This section describes a representation scheme for representing the core knowledge of versatile fault diagnosis discussed in the previous section. In addition, the representation of graphics knowledge necessary for a system to communicate with its users is presented.

3.1 Structural Representation

In our system, a device is abstracted as a hierarchically arranged set of objects, and each object is abstracted at two levels. At level-1 abstraction, an object is modeled as a module with ports; and at level-2 abstraction, the structures of the object is envisioned. An object is represented according to these two abstraction levels from both logical and physical perspectives. Abstractions of an object at these two levels are represented by SNePS rules and SNePS assertions. The former are categorized as *instantiation rules* and the latter as *structural template*. The representation for cross-links between the logical structure and the physical structure is also discussed.

3.1.1 Instantiation Rules as the Level-1 Abstraction

At level-1 abstraction, knowledge about a component type is represented as a SNePS rule. The rule is used later on to instantiate an object of the component type as a module with its own ports and associated functional descriptor. The functional descriptor contains information about the functional description of the component type. The instantiation rule for a physical component type is a little bit simpler in that it contains no functional information of the component type.

The pictorial illustration of the level-1 abstraction of the logical component type "M3A2" is shown in Figure 2. The function of the component type is abstracted as mathematical equations. This is good for digital circuits in general, as well as some simple analog components such as resistors and transformers.

The instantiation rule for objects of the M3A2 type is shown in the SNePSUL (SNePS User Language) command form in Figure 3. The first three lines of the instantiation rule says that "if x is an M3A2-type object, which is a logical object, and it is to be instantiated at its level-1 abstraction (IRfL1), then do the following". The next five "cq's" will instantiate the ports of the object when this rule is executed. I/O ports of an object are the places where the input/output values of the object are stored. Measured (observed) I/O values depict the real behavior of the object, and calculated I/O values show its expected (normal) behavior. The last two "cq's" create the functional descriptors of the object; functional descriptors are pointers to the representation of the function of the object. The first one says "in order to simulate (calculate) the value of the first output, use the function M3A2out1 which takes three parameters, *viz.*, the inputs of the object x in order". The "tolrnc" denotes the tolerance allowed for a measures value when compared to the calculated value. This is especially important for analog components, and is usually set to zero for digital devices.

The pictorial illustration of the level-1 abstraction of the physical component type "MAC3200" is shown in Figure 4. "MAC3200" is the physical equivalent of the logical component type "M3A2". Unlike the M3A2 type, which has five components (or thirteen when the wires are counted as in our design [Taie and Srihari, 1987]), a MAC3200 board has only four chips on it. The structures of M3A2 and MAC3200 will be discussed in more detail later.

The instantiation rule for objects of the MAC3200 type is shown in the SNePSUL command form in Figure 5. The first three lines of the instantiation rule says the "if x is an MAC3200-type object, which is

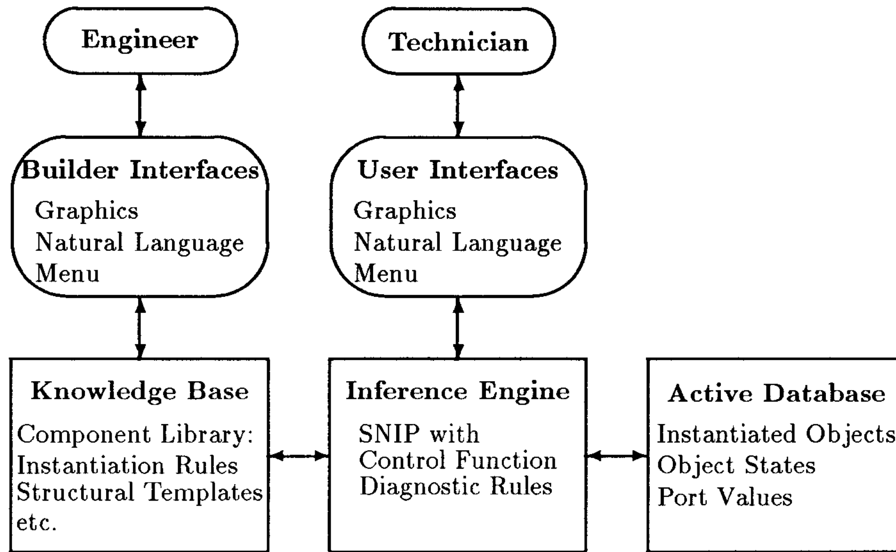
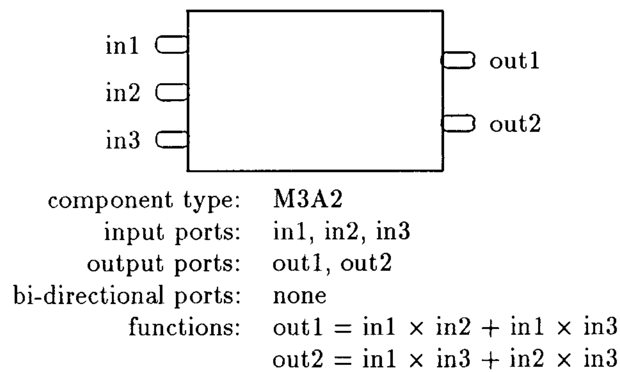


Figure 1: Architecture of VMES



component type:	M3A2
input ports:	in1, in2, in3
output ports:	out1, out2
bi-directional ports:	none
functions:	out1 = in1 $\times$ in2 + in1 $\times$ in3
	out2 = in1 $\times$ in3 + in2 $\times$ in3

Figure 2: The pictorial form of the level-1 abstraction of the logical component type M3A2

```
(build avb $x
       ant (build object *x type M3A2 abs-lv IRfL1 modality logical)
       cq (build modality logical
                 object (build type I-PORT port-of *x id inp1
                               signal (build type D bit-width 2))) = vINP1
       cq (...)
       cq (...)
       cq (...)
       cq (build modality logical
                 object (build type O-PORT port-of *x id out2
                               signal (build type D bit-width 5))) = vOUT2
       cq (build object *vOUT1 sfunc M3A2out1
                 tolrnc 0 pn 3 p1 *vINP1 p2 *vINP2 p3 *vINP3)
       cq (...))
```

Figure 3: The instantiation rule for M3A2 type objects

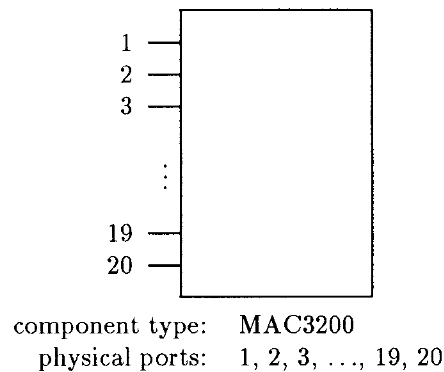


Figure 4: The pictorial form of the level-1 abstraction of the physical component type MAC3200

```
(build avb $x
       ant (build object *x type MAC3200 abs-lv IRfL1 modality physical)
       cq (build modality physical
                 object (build type P-PORT port-of *x id 1))
       cq (build modality physical
                 object (build type P-PORT port-of *x id 2))
       .....
       cq (build modality physical
                 object (build type P-PORT port-of *x id 20)))
```

Figure 5: The instantiation rule for MAC3200 type objects

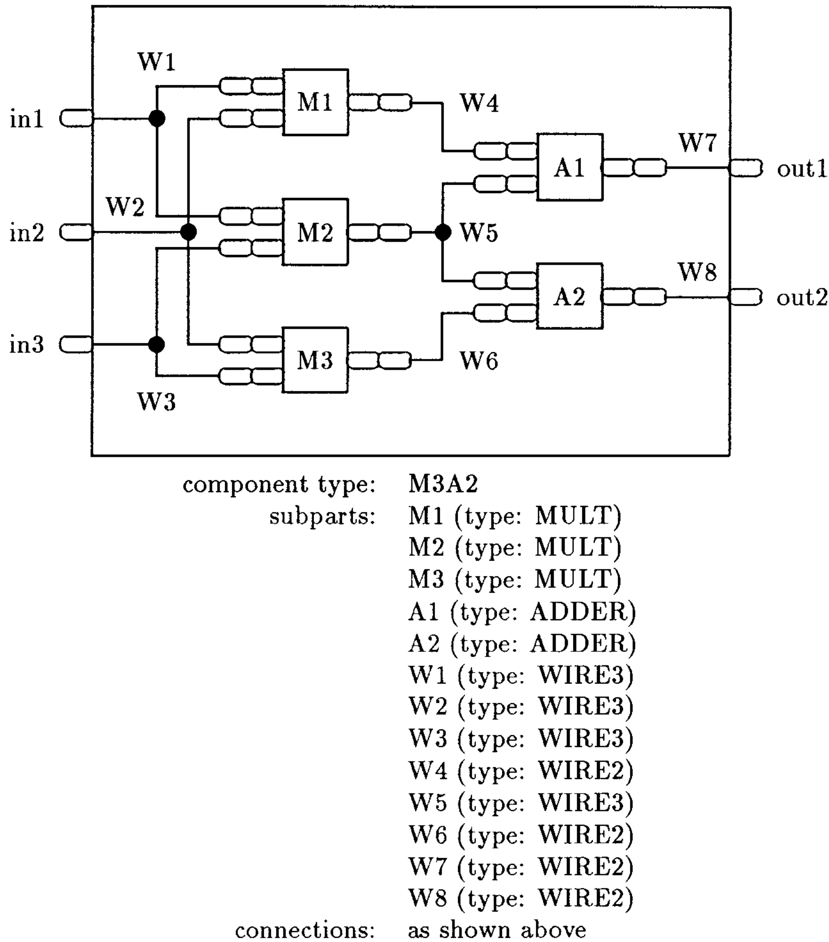


Figure 6: The pictorial form of the level-2 abstraction of the logical component type M3A2

a physical object, and it is to be instantiates at its level-1 abstraction (IRfL1), then do the following". The next twenty "cq's", which are shown in partial to save space, will instantiate the twenty ports of the object when this rule is executed. The instantiation rule for a physical component type is quite similar to that for a logical type component type except that all ports of a physical object are P-PORTs, which are functional (logically) neutral, and thus no functional descriptors are associated with these ports.

3.1.2 Structural Templates as the Level-2 Abstraction

At level-2 abstraction, a structural template, which is implemented as a SNePS assertion, is used to describe the subparts of a logical object at the next hierarchical level, and the wire connections between the object and its subparts, as well as those among the subparts themselves. Since wires are eliminated from the physical abstraction, the structural templates of a physical component type only contain descriptions of its non-wire subparts.

The level-2 abstraction of the logical type M3A2 is shown pictorially in Figure 6. Note that the subparts are partially abstracted at their own level-1 abstractions as modules with I/O ports, but without any functional descriptions. The component types of subparts are also indicated.

The structural template representation is shown in Figure 7. The representation can be viewed as consisting of five parts—an identification section, a subparts section, a connections section, a part-links section. The last two sections in a structural template, whose contents are missing in the above SNePSUL

```
(build

  type M3A2
  abs-lv STfL2
  modality logical

  sub-parts ((build id M3A2-M1 type MULT)
             (build id M3A2-M2 type MULT)
             (build id M3A2-M3 type MULT)
             (build id M3A2-A1 type ADDER)
             (build id M3A2-A2 type ADDER)
             (build id M3A2-W1 type WIRE3)
             (build id M3A2-W2 type WIRE3)
             (build id M3A2-W3 type WIRE3)
             (build id M3A2-W4 type WIRE2)
             (build id M3A2-W5 type WIRE3)
             (build id M3A2-W6 type WIRE2)
             (build id M3A2-W7 type WIRE2)
             (build id M3A2-W8 type WIRE2))

  connections ((build equiv (findorbuild type B-PORT port-of M3A2-W1 id 1
                                    signal (findorbuild type D bit-width 2))
                      equiv (findorbuild type I-PORT port-of M3A2 id inp1
                                    signal (findorbuild type D bit-width 2)))
               .....
               (build contact (findorbuild type B-PORT port-of M3A2-W2 id 2
                                    signal (findorbuild type D bit-width 2))
                      contact (findorbuild type I-PORT port-of M3A2-M1 id inp2
                                    signal (findorbuild type D bit-width 2)))
               .....)

  part-links (.....)

  port-links (.....))
```

Figure 7: The structure template for M3A2 type objects

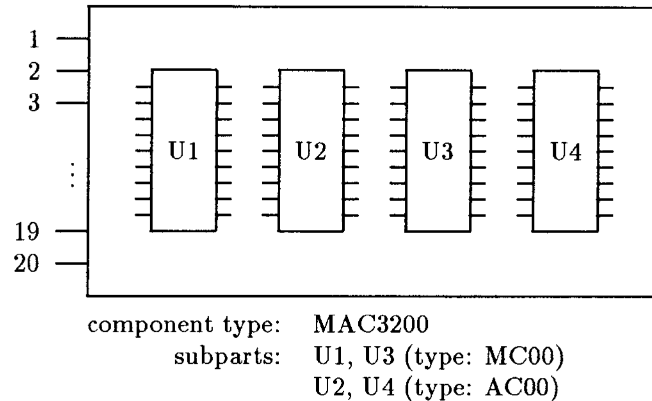


Figure 8: The pictorial form of the level-2 abstraction of the physical component type MAC3200

command, concern the cross-links between the logical structure and the physical structure of a device, and are discussed later.

The identification part, which consists of the first three lines of the SNePSUL "build" command above, denotes that the representation is the structural template for the logical component type M3A2 at the level-2 abstraction (STfL2). The subparts section describes the subparts of the component type at the next hierarchical level. A new case-frame, "id/type", is introduced to describe the subparts of a logical component type within its structural template. The "id" is composed of the name of the component type, i.e., M3A2, and a unique id, such as M3, A1, and W6, within the component type. It identifies the subpart in the rest of the structural template; it also serves for name extension of the subpart when it gets instantiated. For instance, if D1 is an M3A2 device, and its first subpart, which is identified as M3A2-M1 in the structural template, is being instantiated, the subpart will be instantiated with a name of "D1-M1". The part "type" of the subpart specifies its component type; this information is needed when the subpart is to be instantiated. The connections section of the structural template specifies the connections shown in Figure 6. Note that connections by port superimposition and by POCON (point of contact) are treated differently as discussed in [Taie and Srihari, 1987].

A structural template provides the necessary knowledge about the sub-structure of all objects of same component type without representation overhead. Unlike instantiation rules, structural templates are never executed (fired) to produce a representation for any specific object. When reasoning on the sub-structure of an object is required, instead of instantiating the sub-structure (all the subparts and connections) and then reasoning on the resulted representation, we do it directly on the structural template of the object. If suspicious subparts are located, they (but not all subparts) instantiated at their level-1 abstractions using proper instantiation rules for further examination.

The level-2 abstraction of the physical component type MAC3200 is shown pictorially in Figure 8. The subparts are abstracted at their own level-1 abstractions as modules with physical ports. The component types of the subparts are also indicated.

The structural template representation is shown in Figure 9. Unlike the structural template for a logical type, which consists of five sections, the structural template for a physical component type has only two component sections: the initial section and the subparts section. This is because the wires are eliminated from the physical representation of a device, thus no connection is to be specified, and because the cross-links between the logical structure and the physical structure have been specified at the structural template of the logical component type or elsewhere as will be discussed later.

The identification part, which is the first three lines of the SNePSUL "build command", denotes that the representation is the structural template for the physical component type MAC3200 at its level-2 abstraction (STfL2). The subparts section describes the subparts of the component type at the next hierarchical level. A new semantic network case-frame, "id/type/mntn-lv", is introduced to describe the subparts of a physical component type within its structural template. The meaning and use of part "id" and part "type" are

```
(build

  type MAC3200
  abs-lv STfL2
  modality physical

  sub-parts ((build id MAC3200-U1 type MC00 mntn-lv DEPOT)
             (build id MAC3200-U2 type AC00 mntn-lv DEPOT)
             (build id MAC3200-U3 type MC00 mntn-lv DEPOT)
             (build id MAC3200-U4 type AC00 mntn-lv DEPOT)))
```

Figure 9: The structure template for MAC3200 type objects

the same as those in a structural template for a logical component type as described in the last section. The "mntn-lv" indicator shows the intended maintenance level of the subpart, i.e., the maintenance level where the subpart, if found faulty, is replaced without further diagnosis. These informations are used for instantiating a physical subpart.

Knowledge about the intended maintenance level is associated with the physical structure of a device because the repairment of a device is performed based on the physical model of the device. It is adequate to store the "mntn-lv" (maintenance level) tag of an object at the subpart section of the structural template of the object's super-part. A more straight-forward way is to store the "mntn-lv" tag at the instantiation rule of each component type, but this may cause problems. The reason is that "'mntn-lv" values of objects with same component type may be different when they are used in different devices. For instance, an AC00 chip (an adder) may have a different "mntn-lv" value of DEPOT when it is a subpart of Air-Force-Device-1, and have a different value as FIELD when it is a subpart of Navy-Device-3. This implies that the "mntn-lv" value of an object is not only complexity dependent but also environment-sensitive, and thus it should be stored at the subpart section of structural templates rather than at the instantiation rule. Though currently, only FIELD and DEPOT levels are used in VMES, "mntn-lv's" and the corresponding system parameter VMES-IML, which stores the intended maintenance level of a diagnostic session, can be set to any arbitrary maintenance level by the user if desired.

3.1.3 Cross-Links between Logical and Physical Structures

There are two kinds of cross-links between the logical and physical structure of a device. The first kind is the cross-links for components. The second kind is the cross-links for ports. Like representing the level-2 abstraction of a device for its sub-structures, the cross-links between the logical and the physical structures is implemented as structural templates to remove any representation redundancies. The cross-links for components are specified in the part-links section of the structural template of the logical object, and the cross-links for ports are specified in the port-links section as partially shown in Figure 10.

3.2 Functional Representation

The function of an object in the electronic domain can be best abstracted as the relationship between its inputs and outputs as shown in Figure 2. The functional description should be usable to simulate the component behavior, i.e. to calculate the values of output ports if the values of the input ports are given. It should also be usable to infer the the values of the input ports in terms of the values of other I/O ports. This is important if hypothetical reasoning is used for fault diagnosis. Though at this stage, VMES only uses the functional description to calculate values at output ports, our representation scheme can be used both ways.

As depicted by the instantiation rule for M3A2 type, a functional descriptor of a port contains a pointer to its functional description as well as other information concerning the use of the functional description. The functional description itself is implemented as a LISP function which calculates the desired port value

```
(build

  type M3A2
  abs-lv STfL2
  modality logical

  sub-parts (.....)

  connections (.....)

  part-links ((build object M3A2-M1 inside MAC3200-U3)
              (build object M3A2-M2 inside MAC3200-U3)
              (build object M3A2-M3 inside MAC3200-U1)
              (build object M3A2-A1 inside MAC3200-U4)
              (build object M3A2-A2 inside MAC3200-U2))

  port-links ((build equiv (findorbuild type I-PORT port-of M3A2-M1 id inp1
                                  signal (findorbuild type D bit-width 2))
                     equiv (findorbuild
                            bit (findorbuild type P-PORT port-of MAC3200-U3 id 1)
                            lo-bit (findorbuild
                                     bit (findorbuild
                                           type P-PORT port-of MAC3200-U3 id 3))))
  .....))
```

Figure 10: The structure template for cross-links between logical and physical structures of M3A2

```
(defun M3A2out1 (inp1 inp2 inp3)
    (+ (* inp1 inp2) (* inp1 inp3)))

(defun M3A2out2 (inp1 inp2 inp3)
    (+ (* inp1 inp3) (* inp2 inp3)))
```

Figure 11: Output functions for M3A2 type objects

in terms of the values of other ports. Every port of a component type has such a function associated with it. The functional descriptions for the output ports of the component type M3A2 are shown in Figure 11.

Some different ports of different component types might have the same function, some functions can be shared. For instance, the simple function "ECHOback", which simply returns its input, can be shared by several different component types, *viz.*, by the type "super-buffer", the type "wire" and the type "one-to-one transformer". All these component types show the same behavior at out level of component abstraction: they echo the input to the output.

3.3 Graphics Knowledge Representation

While the process of diagnosis is running, the user is informed about the activities of the system via a graphical trace of its reasoning. The diagnosed device is displayed on a graphics terminal, and parts currently under consideration are highlighted, for instance, by changing their colors. All the information necessary to create a graphical representation of the diagnosed object is stored in the very same knowledge representation environment (SNePS). This not only means that we are using the same SNePSUL syntax to describe objects in a way that pictures can be created, but we are also using a common knowledge base, and in fact to a certain degree the *same knowledge* for the the diagnosis and the drawing programs.

It is necessary to know about the *form* of every object involved in the production of a drawing. In VMES, forms are either linked directly to the corresponding object or an object inherits a form from a class of objects. This requires two case frames, one linking the object to a class and a second one linking the class to a form. Forms represent the link between the declarative and the procedural plane of the representation system. A form is at the same time two different things: it is a (base) node in the semantic network, and in this way accessible by the knowledge base handler, but it is also the name of a LISP function that contains calls to routines of a LISP graphics package. SNePSUL expressions for some of the case frames used in our system are given in Figure 12.

4 Diagnostic Reasoning

4.1 Control Structure

Human problem solving performance in diagnostic tasks has been studies using experiments by many researchers in the field of psychology [Bond, Jr. and Rigney, 1966; Dale, 1957; Hunt and Rouse, 1984; Rasmussen and Jensen, 1974; Rouse, 1978a; Rouse, 1978b; Rouse, 1979b; Rouse, 1979a; Rouse *et al.*, 1980; Rouse, 1984]. For a specific type of networks, which consist solely of AND gates (called *nodes*) and are tested with a 1 on every primary input, an optimal topological search procedure based on the half-split principle and single fault assumption is proposed [Rouse, 1978a]. A slightly revised version of this procedure is summarized in Figure 13. Experiment results indicate that human performance deviates from the optimal half-split procedure as problem size increases due to human cognitive limitation [Rouse, 1978a]. However, people can be trained to better use structural information in diagnosis.

The half-split procedure described in Figure 13 can be easily extended to handle general logic networks. Given a device with its symptom, the expected output values of all its components with respect to the input vector are computed using the interconnections and functions of the components. It is also noticed that a 0 in a network of AND gates with all primary inputs being 1's generalizes naturally to a violation between

```
; This describes an object with Individual Form
(build object D1-M1
       form xmult
       modality function)

; This asserts that D1-M2 is a multiplier
(build object D1-M2
       type multiplier
       modality function)

; This expression links the class multiplier to the form xmult
(build class multiplier
       form xmult
       modality function)

; This is a partial description of D1.
; It has two parts D1-M1 and D1-M2, one sub-assembly
; which is an input port with the id inp1 and an
; absolute position at (100, 200)
(build object D1
       subparts (D1-M1 D1-M2)
       sub-assems (build inport-of D1 id inp1)
       abs-pos (build x 100 y 200)
       modality function)
```

Figure 12: Example case frames for graphics knowledge

procedure OPTIMAL SEARCH
  Form the feasible set (FS) of nodes
  which reach all known 0 outputs but no known 1 outputs
  while $|FS| > 1$ **do**
    Select an arc such that $|RS|$ is closest to $|FS|/2$,
    where $RS \subseteq FS$ is the set of nodes that reach the arc
    if the value on the arc is 0
      then Reduce FS to RS
      else Remove RS from FS
    endif
  endwhile
endprocedure

Figure 13: An optimal topological search

```
procedure diagnose (cl: an ordered candidate list)
  while there are unexplained violations do
    Instantiate the first candidate at its level-1 abstraction
    Measure its inputs and outputs
    if it has violated outputs then
      if its corresponding physical object is at IML then
        Issue repair order for the physical object
      else
        Instantiate it at its level-2 abstraction
        Generate and order suspected components of it using its structural description
        Call diagnose on the ordered suspected components
      endif
    else
      Claim that the current candidate is intact
    endif
    Propagate measurements to update predications
    Update candidate list according to reordering and elimination principles
  endwhile
  Report findings
endprocedure
```

Figure 14: Control structure of VMES

expected and actual values in a general logic network, while a 1 generalizes to a corroboration. With these modifications, the procedure can now be used for general networks.

For this guided probe procedure to handle multiple faults, principles of candidate ordering, reordering and elimination are developed [Chen and Srihari, 1989]. Initially, components are ordered using structure and symptom information. A component which connects to more incorrect primary outputs is considered more likely to be faulty. This is because we want to use as few faulty components ars possible to explain the observed discrepancies, also known as "Occam's razor" or the principle of parsimony [Reggia *et al.*, 1983; Reggia *et al.*, 1985]. In case of a tie, relationships with corroborations are used to break the tie—a component connecting to more correct primary outputs is considered less likely to be faulty.

The initial ordering represents our estimation of the relative component fault probabilities. However, these relative probabilities may change after new measurements are made. This gives rise to the following reordering principle. Whenever a violation is found, candidates connecting to the violation are moved to the front of the candidate list. Sometimes, a component may even be exonerated after a measurement. This is captured in the candidate elimination rule—whenever a corroboration is found, candidates connecting to the corroboration but not to any violations without passing through a corroboration are removed from the candidate list. The elimination principle is stated with single fault in mind. In multiple fault cases, those components can not simply be removed because a corroboration may be caused by two faults which cancel each other's fault effects. However, they can still be moved to the end of the candidate list as a heuristic.

Figure 14 shows the diagnostic procedure of VMES using these ordering principles. It starts from the top level of the structural hierarchy of the diagnosed device by instantiating the device at its level-l abstraction. It then tries to find the device's output ports that violate their expectations (i.e., output ports that have different observed values from expected ones). A candidate is claimed to be intact (with respect to the current inputs) if it has no violated outputs.

After detecting violated outputs of the current candidate, the system determines if it is necessary to examine the components of the candidate based on the idea of "intended maintenance level" (IML). The candidate is declared faulty and a repair plan is formed for its corresponding physical object if the physical object is at the intended maintenance level selected by the user at the beginning of the diagnosis session.

Otherwise, the candidate is instantiated at its level-2 abstraction. The structural description is then used to find, at the next lower hierarchical level, a subset of its components which might be responsible for the violated outputs of the current candidate. These suspected components are ordered according to some ordering criteria which will be described in detail in the next section. This diagnosis process is then recursively called for the new ordered components.

After the current candidate is checked, its input and output measurements are propagated toward the primary outputs of its super-part at the next higher hierarchical level so that the predictions, as well as violations and corroborations, are up-to-date. The remaining candidate list is then updated according to the reordering and elimination principles.

It remains to determine when a diagnosis should be terminated in multiple fault cases. For single fault cases, a diagnosis can be terminated as soon as the first fault is found. For multiple fault cases, a diagnosis continues until all violations are explained by the known faults. In VMES, the intended maintenance level is also used to shortcut a diagnostic session. When all the remaining candidates are in a single physical unit whose maintenance level is the same as the intended one, diagnosis is terminated without further distinguishing which parts are actually at fault since the same physical unit will be replaced anyway.

To show that the reordering and elimination principles really help, the performance of this procedure is analyzed based on the number of components which are checked during diagnosis. Let $len(n, k)$ denote the average length of a diagnosis (number of components checked) to find the first k faults when there are n *active candidates.* For simplicity, we assume the diagnosed device has only two levels, i.e., its components have no subpart. Active candidates comprise the beginning section of the candidate list and contain at least one faulty component if there is one in the candidate list. Initially all components are active candidates and the active candidate list is always shrinking until a fault is found. The remaining candidate list becomes the active candidate list as a fault is found, When the candidate list is reordered, the active candidates which are moved to the front become the new active candidates. When candidate elimination is applied, the active candidates which are not eliminated (or are not moved to the end) become the new active candidates. Then $len(n, 1)$, the average length of a diagnosis for finding the first fault is given by the following recurrence relation:

$$\begin{aligned} len(n,1) &= \frac{1}{n} \cdot 1 \\ &\quad + \frac{n-1}{n} \cdot [1 + \sum_{i=1}^{n-1} \frac{1}{n-1} \cdot len(i,1)]. \end{aligned}$$

The first term represents the case that the first candidate is faulty (with probability $1/n$) and only one component is checked. If the first candidate is intact (with probability $(n-1)/n$), 1 to $n-1$ active candidates are left and each case has an equal probability of $1/(n-1)$. The average number of checked components for this case is computed by the second term. The above expression simplifies to

$$\begin{aligned} len(n,1) &= \frac{1}{n} + len(n-1,1) \\ &= \frac{1}{n} + \frac{1}{n-1} + len(n-2,1) \\ &\vdots \\ &= \frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2} + 1 \\ &= \sum_{i=1}^{n} \frac{1}{i} \\ &= \log n. \end{aligned}$$

To derive $len(n, 2)$, let $p_i (\geq 0)$ be the probability that the first fault is found at the ith time ($i = 1, \ldots, n$). Then, $\sum_{i=1}^{n} p_i = 1$ by the definition of probability and $\sum_{i=1}^{n} p_i \cdot i = len(n, 1) = \log n$ according to the analysis

for $len(n,1)$. Now $len(n,2)$ can be described as follows:

$$len(n,2) = \sum_{i=1}^{n} p_i \cdot [i + len(n-i,1)],$$

where $[i + len(n-i,1)]$ is the average checked components when the first fault is found at the ith attempt and p_i is the probability of this case. Since $len(i,1) = \log i$,

$$\begin{aligned} len(n,2) &= \sum_{i=1}^{n} p_i \cdot i + \sum_{i=1}^{n} p_i \cdot \log(n-i) \\ &= \log n + \sum_{i=1}^{n} p_i \cdot \log(n-i) \\ &\leq \log n + \sum_{i=1}^{n} p_i \cdot \log n \\ &= \log n + \log n \sum_{i=1}^{n} p_i \\ &= \log n + (\log n) \cdot 1 \\ &= 2 \log n. \end{aligned}$$

By mathematical induction, we have

$$len(n,k) \leq k \log n,$$

where $1 \leq k \leq n$. This means that the length of diagnosis, when the number of faults is relatively small (which is true for most real diagnostic problems), grows logarithmically with respect to the number of components. In particular, for single fault cases, the average number of checked components is $\log n$ which is comparable to the performance ($\log_2 n$) of the optimal half-split procedure and is much better than that of random or sequential examination whose expected length is $n/2$.

4.2 Diagnosing Wires and POCONs

All wires have the same function, i.e., transmitting values (signal as voltage) from one point to another. Though wires may have different number of wire ends, they all show the same behavior—the values at all ends of an intact wire are equal. It is usually necessary to simulate the behavior of a common component by calculating its outputs from its inputs using its functional description [Davis, 1984; Genesereth, 1984; Shapiro *et al.*, 1986; Taie and Srihari, 1986]. Unlike common components, there is no need to simulate the behavior of a wire. A simple rule which states that a wire is faulty if it has different values at its ends suffices. The rule we use to diagnose wires is shown in Figure 15.

One interesting point of this rule is that it neither specifies that p1 and p2 are two different ports in the antecedent part of the rule, nor does it specify that v1 and v2 are different anywhere. The reason is that, under the UVBR (Unique Variable Binding Rule) of SNePS [Shapiro, 1986], different variables cannot bind to the same value. This has the advantage of better reasoning efficiency by saving some antecedents of a rule, and by eliminating redundant variable bindings.

A POCON (point of contact) has a similar function to a wire—it transmits a signal from one side to the other. Remember that our model for POCON is a logical object which is represented as a structured node with two "contact" arcs pointing to two ports of two two components. Therefore, a bad POCON can be defined as a POCON whose two associating ports have different values. Also note that a POCON is only a conceptual object which represents a physical relationship between two ports of two components. It has no port of its own, and whenever the value of a port of a POCON is requested, the port is identified as a port associating with its host component rather then the POCON.

In VMES, the locating of a bad POCON is treated as a byproduct of checking components. Whenever a port value is acquired, the system finds another port which forms a POCON with the first port. If the

```
; IF p1 and p2 are (different) bi-ports of wire w, and
;    v1 and v2 are the measured values of p1 and p2, and
;    v1 is not equal to v2
; THEN wire w is faulty
(build avb ($p1 $p2 $w $v1 $v2)
       &ant ((build object *p1 bi-port-of *w)
             (build object *p2 bi-port-of *w)
             (build object *w type WIRE)
             (build object *p1 attr (build atrb-cls M-value atrb *v1))
             (build object *p2 attr (build atrb-cls M-value atrb *v2)))
       cq (build *w attr (build atrb-cls state atrb faulty)))
```

Figure 15: A rule for diagnosing wires

second port has a value which is different from the value of the first port, then the system concludes that the POCON is bad. This is somewhat similar to the way a human expert diagnoses electronic circuits in that he measures some port values to check a component, but the result of the measurement stimulates him to conclude that fault is on a nearby POCON rather that on the components he intended to check [Rasmussen and Jensen, 1974].

4.3 Repair Suggestion

At the end of a diagnostic session, VMES suggests a repair plan to the user according to the type of the faulty object. If the faulty object is a common component, VMES simply suggests the user to replace its physical corresponding part (Figure 16a). If it is a wire, the physical corresponding wires are identified for repair (Figure 16b). Note that a logical wire may correspond to several physical wires, e.g., a 4-bit logical wire is realized by four wires on a printed circuits board. If the faulty object is a POCON, the user is directed to the location of the contact point (Figure 16c).

5 Intelligent User Interface

VMES contains a knowledge based graphics package which is used as part of the user interface. New designs for user interfaces and "Graphical Deep Knowledge" are investigated. We consider a knowledge representation system to be dealing with Graphical Deep Knowledge (as opposed to graphical knowledge), if the knowledge is organized in a way that makes it accessible not only to display routines, but also supports some form of graphical reasoning with this knowledge. The theory for Graphical Deep Knowledge is discussed in detail in [Geller, 1988]. Part of the developed theory has been implemented as a generator program (**TINA**) that creates pictures from knowledge structures. It has also been implemented as a creation program (Readform) and as an ATN grammar that create knowledge structures from menu oriented input and limited natural language input, respectively.

5.1 The TINA Graphics Interface

The use of the **TINA** program [Shapiro *et al.*, 1986; Taie *et al.*, 1987; Geller *et al.*, 1987; Taie, 1987] as a graphics interface of the VMES project is described in this section. The VMES system consists of a maintenance reasoner and a graphics interface. The graphics interface is an application of the **TINA** program. The task of the maintenance reasoner is to identify a faulty component in a given device, usually a circuit board. The maintenance reasoner and the display program share a knowledge base realized as a SNePS network.

During the process of identifying a faulty component in a device, the maintenance reasoner repeatedly updates the shared knowledge base. It categorizes components as being in a "default state", being in a

```
>>>>>> I GOT THE FAULTY PARTS AS >>>>>>
(D1-M2)
$$ Repair Order: replace D1-U3 (type MC00)
```

(a) For a common component

```
>>>>>> I GOT THE FAULTY PARTS AS >>>>>>
(D1-W1)
$$ Repair Order: fix the wire connecting
pin 4 of D1
pin 8 of D1-U3
pin 4 of D1-U3
$ and also the wire connecting
pin 3 of D1
pin 10 of D1-U3
pin 2 of D1-U3
```

(b) For a wire

```
>>>>>> I GOT THE FAULTY PARTS AS >>>>>>
The POCON of port 3 of D1-W1 and port inp1 of of D1-M2
$$ Repair Order: fix the contact point at
pin 10 of D1-U3
```

(c) For a POCON

Figure 16: Repair suggestions made by VMES

state of violated expectation, being recognized faulty or being suspected to be faulty. Information about any of these states is asserted in the network, using the attribute case frame described earlier on. Whenever the maintenance reasoner wants to express changes in its state of knowledge about the analyzed device, it executes a call to **TINA**. **TINA** presents the current state of the maintenance process to the user. This is done by mapping attributes into signal colors (red = faulty, blue = default, green = suspect, magenta = violated expectation).

Typically a device will be displayed completely blue in the beginning. After finding a violated expectation, for instance a port that has a wrong voltage value, this port will receive an attribute "violated expectation". The device will now be blue, except for the port in question which will be magenta. Finally, after several steps of reasoning and redisplay, the device will be shown in blue with the faulty component(s) in red.

The procedural interface between maintenance reasoner and display program consists of the **TINA** function only! All other communication is done through the shared knowledge base that both parts of the program have access to. Our experience with this type of programming has been that it is exceedingly easy to combine two independently developed modules.

5.2 The Readform Interface for Object Creation

The second interface is the "Readform" program which is used for the creation of visual icons in a format that is accessible to the knowledge representation system. This avoids the necessity of hand generation of graphic code. The compilation of a larger pictorial unit is done by asserting information about objects in the network, such that in the process of drawing access is made to the icons created by Readform. Readform works menu oriented and permits the user to create an object of lines, polygons, circles, disks, boxes, blocks (filled boxed), arcs and text.

By observing users in the process of object creation, it has become obvious that the internal conceptual structures of the person can to a certain degree be derived from the order of his actions as well as by asking a few questions at strategic points. Readform supplies the user with a scratch buffer which is separate from the object created at the current moment. Users have been observed to create objects by drawing a simple unit in the scratch buffer and then repeatedly yanking the buffer content into the picture.

From this chain of actions one can derive that all the yanked objects are presumably members of a certain class, and the system can verify this by asking the user whether there is in fact such a class, and if so, how to name it. This information can be used to create exactly the Graphical Deep Knowledge Structures that are used for picture generation.

The other thing that can be derived from the above chain of user interactions is that all the yanked icons are presumably parts of a larger object which consists of all the iconic primitives (lines, arcs, etc.) which were not created by using the scratch buffer. This permits a system to ask the user whether he wishes to name this larger object separately, and if he desires so, a part relation between the yanked parts and the main structure can be formulated and stored in the knowledge base as a proposition. This proposition becomes part of the part hierarchy in the knowledge base. Part hierarchies are a backbone of many representational systems and are used in the process of maintenance reasoning as well as having major importance in the derivation of pictures from Graphical Deep Knowledge and in controlling complexity of displayable pictures.

5.3 The Natural Language Interface

The third user interface of VMES is the natural language interface. A versatile maintenance system is in need of a user interface in two different situations. In the first situation a maintenance technician uses the system to get help in troubleshooting a currently faulty device. The second situation is as important, namely the initial creation of the device representation. In order to deserve the title "versatile", it must be possible to create device representations with ease and flexibility. The natural language interface that will be described here belongs to the second class of interfaces. It is the goal of the interface to create an internal device representation to the point where it is possible to display the whole device. However, as much of the creation as possible should be done with natural language.

For the purpose of drawing logical circuit diagrams, referred to as Intelligent Machine Drafting (IMD) [Geller, 1988], there is no necessity to actually enter coordinate information, so the natural language descriptions become quite natural. Natural language processing is done by way of an ATN interpreter/compiler

that is part of the SNePS environment [Shapiro, 1982]. The class of objects that can be built by natural language is limited. The major limitation imposed by natural language interface is the branching factor of electrical connections. It is possible to create wires impinging on at most three ports.

In Figure 17, the original set of sentences that is understood by the natural language interface and that describes the Adder-Multiplier circuit is presented. Running this set of sentences through the ATN interpreter will create all the structures necessary to describe the circuit completely for display purposes.

The first `(nl)` in Figure 17 calls the natural language processor from the SNePS environment, while the `^end` at the end returns to the SNePS environment. Although the vocabulary of this interface is quite limited there are variations of the sentences shown above possible.

Of special interest are the final sentences that starts with "the form" because these sentences call, if necessary, the aforementioned Readform interface from inside the ATN interpreter and not only assert the relations between object class and form, but also create any unknown form-icons by having the user drawing this icon. If the form is already known to the system, then only the assertional component of this operation will be executed. The device created by these sentences is shown in Figure 18.

It is important to assert that IMD differs from Computer Aided Design (CAD) in that it deals with functional representations as opposed to structural representations and that the goal of solving a layout and routing problem is to create a "readable" and ideally even "appealing" functional design, as opposed to an optimized structural design.

6 Conclusion

In diagnostic problem solving, human experts seem to use both the logical structure and the physical structure of the target device throughout the diagnostic process at every hierarchical level. Knowledge of the logical structure of the target device together with the associating functional knowledge is used for diagnostic reasoning, and knowledge of its physical structure is used to carry out a test, to determine when a diagnostic session can be terminated, and to form a repair plan. It is important to incorporate the physical representation and the logical representation of a device in maintenance. We find that a physical representation of the the target device, together with the representation of the cross-links between the logical and physical structures of the device, contributes to fault diagnosis in several aspects. It helps determine when a diagnostic session should be terminated, thus it provides versatility across maintenance levels. It can provide a shortcut to diagnosis by noticing that all logical candidates are in a physical object at the intended maintenance level. It helps to form a repair plan based on the physical nature of the target device. Finally, physical representation eases user interaction by directing the user to the exact location in the real device for test and repair.

The most important feature of VMES is its versatility. VMES can easily be adapted to new devices by simply adding the structural and functional information of the new component types to the component library. A new component type is a component type that has not previously been described in the component library. The effort required to adapt the system to new devices should be minimal since digital circuit devices have a lot of common components, and the structural and functional description are readily available at the time a device is designed. As another dimension of the versatility, the diagnostic reasoning procedure of VMES does not require the single fault assumption. Yet it is effective for a small number of faults and is comparable to the optimal half-split procedure for single fault cases.

We have also found that the graphics interface considerably improves the understandability of the reasoning process of the system during diagnosis. The use of a knowledge based graphics system promises to simplify the creation of graphics for new devices, in this way adding the versatility of the system. The common representation for diagnosis, graphics and a number of natural language tools has aided us in adding a natural language component to the system, in this way strengthening our belief in the usefulness of a knowledge based graphics system as a natural interface for a user friendly maintenance expert system.

```
(nl)
D1 is a board
D1M1 is a multiplier
D1M2 is a multiplier
D1M3 is a multiplier
D1A1 is an adder
D1A2 is an adder
D1 has 3 inports
D1 has 2 outports
D1M1 has 2 inports
D1M1 has 1 outport
D1M2 has 2 inports
D1M2 has 1 outport
D1M3 has 2 inports
D1M3 has 1 outport
D1A1 has 2 inports
D1A1 has 1 outport
D1A2 has 2 inports
D1A2 has 1 outport
connect input 1 of D1 with input 1 of D1M1 and input 1 of D1M2
connect input 2 of D1 with input 2 of D1M1 and input 1 of D1M3
connect input 3 of D1 with input 2 of D1M2 and input 2 of D1M3
connect output 1 of D1M1 with input 1 of D1A1
connect output 1 of D1M2 with input 2 of D1A1 and input 1 of D1A2
connect output 1 of D1M3 with input 2 of D1A2
connect output 1 of D1A1 with output 1 of D1
connect output 1 of D1A2 with output 2 of D1
D1M1, D1M2, D1M3, D1A1, and D1A2 are parts of D1
wires are parts of D1
the form of a board is xboard2
the form of a multiplier is xmult2
the form of an adder is xadd2
the form of a PORT is xport
show D1
^end
```

Figure 17: Sentences for creating a circuit

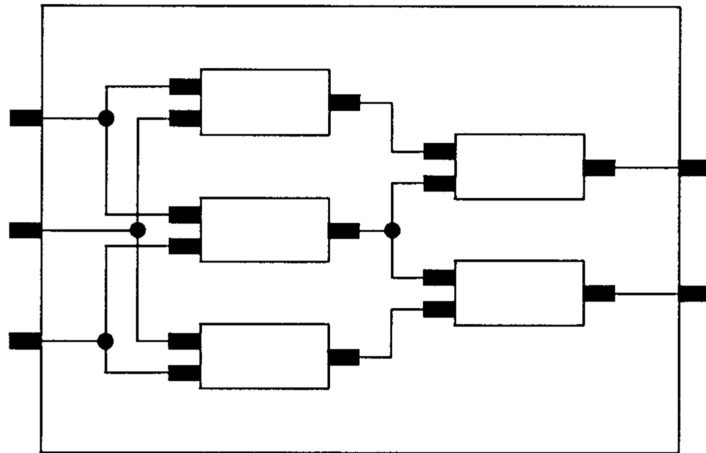


Figure 18: A device created using natural language interface

A Representation and Diagnosis of Sequential Circuits

A.1 Introduction

Constraint based representation of structure and behavior has been traditionally used to localize faults in combinational circuits [Davis, 1983]. This scheme was extended to represent sequential circuits, with the use of layers of temporal granularity and a vocabulary of signals appropriate to the circuit [Hamscher and Davis, 1984]. Expectation violation has been used for candidate generation in combinational circuits. The same procedure, when applied to sequential circuits, was found to be indiscriminate. Single stepping was suggested as a divide and conquer strategy to localize faults in sequential circuits. Structural detail was however proposed to make candidate generator discriminating [Hamscher and Davis, 1984].

Here, we present the details of using structural detail to diagnose sequential devices. We outline candidate generation based on electrical behavior, using fault characteristics which conveniently express structural details. We detail representation and handling of sequential circuits, its unique problems and theoretical / technical solutions. Further, we sketch the diagnostic steps necessary for sequential devices, over and above those used for combinational devices. Finally, we talk about assumptions, and how they should be relaxed in a multiple-symptom case, so as to make diagnosis efficient and correct.

A.2 Representation of Sequential Circuits

Hierarchical representation of circuits aids proper focusing of diagnosis. It makes details available on an as-necessary basis, so that there is neither glut nor dearth of information during diagnosis. Therefore, the structure of circuits has been represented in layers of hierarchy.

Sequential circuits are admittedly more complex than combinational circuits because they have an added time dimension. Their behavior may vary with time because of in-built memory. To conveniently model sequential circuits, this time factor must be taken into account. Therefore, we proceeded to propose a temporal hierarchy for sequential circuits.

The following algorithm lays the rules for temporal hierarchy among circuits. It starts with the most basic blocks and builds layer by layer, up to the most complex circuits.

(1) Basic gates are represented at two levels: the gate delay level which is necessary for analysis of faulty devices causing racing conditions in circuits; and at the input-output relation level. This maybe truth-table or boolean expression. For most purposes the second representation will be adequate. Moreover, since VMES does not deal with parametric faults, the first representation has been ignored in the current VMES system.

(2) Flip-flops are represented at two levels: the clock period level and the overall behavior level. The overall level may be transition diagram, state table or just a function.

(3) Any complex unit such as a chip, which is at the basic maintenance level is represented as in (1) or (2) depending on whether it is sequential or combinational.

(4) Complex modules such as boards are represented at two levels, if possible: at the level lowest among the highest levels of representation of the immediate sub-modules; and at the overall behavior level of the board.

The different levels of time representation chosen are however expressible as integral multiples of the more basic levels. Also note that the two levels of representation suggested may be the same for some devices. This scheme is applicable to most general cases of synchronous circuits.

A.3 Handling Feedback in Sequential Circuits

Sequential circuits are made up of sequential and combinational components or subdevices. These components interact closely to constitute the behavior of the whole circuit. By virtue of this close interaction, subdevices of a sequential circuit are better addressed from the perspective of the encompassing superdevice. As we will see along this section, this applies to the representations of function and fault characteristics and handling of states of components in a sequential circuit. The terms 'circuit' and 'superdevice' are used interchangeably in the following discussion, as are the terms 'component' and 'subdevice'.

Sequential circuits have feedback, also termed memory. Memory is dealt with as the state of the device. Output of a sequential device depends not only on the inputs but also on its state. The state of a device is in

turn, a function of its outputs in the previous clock cycle. Hence, representation of a sequential component should provide for the storage of the state of the device. This storage variable should be initialized at the beginning, and re-assigned after every clock cycle.

Initialization of a circuit involves initializing all the sequential subdevices constituting the circuit. Initialization of a component consists of setting its state variable(s) to a value. This value is either a fixed number or a function of some inputs of the superdevice. Initialization routines for subdevices are stored with the superdevice and are executed upon entry into the superdevice.

Functions of sequential components are expressed at the lower level of temporal hierarchy of the encompassing superdevice. For instance, if 8 bit adder is a subdevice in a sequential multiplier, and the temporal hierarchy of the sequential multiplier consists of (1) functional level : out1 = in1 * in2 and (2) clock cycle level, the function of the 8 bit adder is expressed per clock cycle. During simulation of the superdevice, the functions of the subdevices are executed in topological order as many times as the number of clock cycles in the superdevice. We note that application of function in sequential components involves states besides input and output ports. The value of state may be used as an input or the state variable maybe treated as on output for the component function.

Due to the necessity of feedback, sequential circuits usually have closed loops in their topology. However, to carry out simulation and reasoning step by step, (at the relevant temporal unit) these loops must be severed appropriately to provide pseudo-input and pseudo-output ports. In other words, it is necessary to recognize the subdevice(s) that store the state of the superdevice, and count the inputs and outputs of these subdevices as pseudo-outputs and pseudo-inputs of the superdevice respectively. *Loopbreaks* are the pseudo-inputs of the superdevice that carry the previous state of the superdevice for purposes of function application. *Loopbreak-heads* are the subdevices whose outputs are the pseudo-inputs of the superdevice. During simulation of the superdevice behavior, the following steps are carried out in order:

(1) The superdevice and its relevant sequential subdevices are initialized

(2) First time around, starting with primary inputs of the superdevice and outputs of the *loopbreak-heads*, functions of every subdevice is executed in topological order. (This order may not be strict and may involve backtracking) The topological order stops either at primary outputs or after *loopbreak-heads.*

(3) On subsequent rounds, one of two procedures may be followed: If the superdevice takes in a fresh set of inputs on every round, procedure (2) is repeated. If however, the superdevice works on only one set of inputs (i.e., those taken in on the first round) procedure (2) is carried out with the outputs of the subdevices connected to the primary inputs of the superdevice substituting for the primary inputs of the superdevice. Step (3) is repeated for as many times as the function of the superdevice warrants. For instance, for the 4 bit sequential multiplier that takes 4 clock cycles to calculate the product, each of the above rounds may be taken as one clock cycle. Thus, representation and simulation of sequential circuits is considerably different from that of combinational circuits because of the inherent feedback.

A.4 Candidate Generation based on Electrical Behavior

Sequential circuit diagnosis is admittedly hard. Therefore, every opportunity to narrow down the list of possibly faulty devices should be exploited to the maximum. With this goal in mind, we propose candidate generation based on electrical behavior.

Candidate generation based on electrical behavior rests on the following observation: Not all the inputs of a component will be responsible for some observed wrong value at its output; Further, many of these inputs that are not responsible for the faulty output can be identified and eliminated from further consideration with full certainty. The knowledge, which inputs are not responsible, or in other words, which inputs may be responsible for an observed wrong value at the output of a component, is expressed as the fault characteristics of the component. For complex devices, i.e., superdevices, fault characteristics also capture the list of subdevices that are possibly faulty, given some faulty values at the outputs of the superdevices. In short, fault characteristics are indicators of culpability. For example, fault characteristics for an AND gate are as follows. In the table, output values are measured, input values are expected and the last column specifies the inputs to suspect and to propagate backwards from.

			AND gate Fault Characteristics
OUT1	IN1	IN2	SUSPECT / PROPAGATE
0	1	1	either in1 or in2 or both
1	1	0	input supposed to be 0 : in2
1	0	1	input supposed to be 0 : in1
1	0	0	none if single fault assumption unless inputs tied

Following are some note-worthy observations about fault characteristics:

(1) Knowledge of the value of the faulty output is necessary to apply fault characteristics. The characteristics vary with the faulty value. Moreover, characteristics are particular to the output of the component, and change from one output to another, unless the function (and in cases, the internal structure) of the outputs are the same.

(2) Depending on the nature of the device, fault characteristics are helpful to varying degrees. For example, for the AND gate, they are helpful in all cases except when both inputs are 1. However, for the exclusive OR gate, as seen below, they are not of much help except to point out that only one input could be wrong in any given case.

			Ex-OR gate Fault Characteristics
OUT1	IN1	IN2	SUSPECT / PROPAGATE
1	0	0	either in1 -> 1 or in2 -> 1
1	1	1	either in1 -> 0 or in2 -> 0
0	0	1	either in1 -> 1 or in2 -> 0
0	1	0	either in1 -> 0 or in2 -> 1

(3) For all simple devices, fault characteristics could be pre-computed and pre-compiled. With this information readily available, diagnosis can be sped up considerably.

(4) Since sequential circuits have closely interacting components, fault characteristics can less apply to individual components than to complex superdevices. The fault characteristics of a superdevice specify which subdevices to suspect, given some violation at the outputs of the superdevice. Again, these characteristics are expressed at the higher level of temporal representation of the superdevice. The lower level of representation is typically utilized for candidate elimination. For example, for the 4 bit sequential multiplier, fault characteristics are expressed in terms of the possible products that may appear at the outputs of the multiplier. However, user is asked for measurements at the clock cycle level, so that measured and computed values can be compared for candidate elimination.

(5) Fault Characteristics capture the internal structure of devices, especially for sequential devices. They introduce structural information into sequential circuit diagnosis, thereby making the task more feasible. Therefore, not only the function but also the internal structure of a component is required to generate fault characteristics for the component.

Fault characteristics can be easily formalized for only the simpler devices that reside at the lowest level of structural hierarchy. For more complex devices, it is hard, even infeasible and unnecessary to have pre-computed fault characteristics. Instead, fault characteristics for the device, for the observed faulty output, is computed on the run from the characteristics of the subdevices, and as part of the diagnostic procedure. This procedure is as follows:

```
FOR each faulty output of the device
    get all the devices connected to it
    FOR each device connected to the output
        get its fault characteristics
        find all the inputs responsible for the faulty output of the device
        recurse on each of the inputs
    END-FOR
END-FOR
```

The fault characteristics so computed simply constitute the list of candidates that need to be checked. The above procedure could be tuned to further narrow down this list by incorporating forward reasoning in case of single fault assumption. In essence, this means that when a new subdevice is added to the list,

the algorithm reasons forward from the subdevice towards the primary outputs of the superdevice. If this reasoning ends at any good output of the superdevice, then the subdevice can be eliminated from the list right away. This procedure is based on the observation that a bad subdevice output cannot contribute to a good superdevice output when single fault assumption is made. This has not been built into the system VMES yet.

Candidate generation based on electrical behavior is significantly different from candidate generation based on topology [Chen and Srihari, 1989]. In the topological procedure, only the knowledge of whether an output is faulty or not suffices for candidate generation. However, in the electrical procedure, we will also need the details as to how the output is faulty (i.e., measured values). In diagnosis, it is reasonable to assume the availability of this information. Topological procedure is heuristic, whereas electrical procedure is algorithmic. The advantages and disadvantages of heuristics versus algorithms mostly apply to the comparison of the two procedures as well. Whereas topological procedure works fine without any knowledge of the functions of the subdevices in the circuit, electrical behavior cannot do without the information. Since electrical procedure utilizes more information about the circuit in question, it can be expected to perform at least as well and possibly better than the topological procedure. However, the associated cost (in terms of extracting fault behavior, and using electrical behavior for candidate generation) is also higher.

A.5 Diagnosis of Sequential Circuits

The following control structure of VMES was adequate to diagnose combinational circuits:

```
REPEAT
    Simulate the circuit at the next structural level
    Single step once down the structural hierarchy
    Generate candidates by constraint elimination
    Eliminate candidates by asking for more information
UNTIL circuit diagnosed or basic level of structural hierarchy reached.
```

However, Sequential circuits have temporal complexity in addition to the structural complexity found in combinational circuits. Therefore, temporal hierarchy was proposed earlier for sequential circuits. The control structure had to be suitably modified to handle this hierarchy and exploit its advantages. The new control structure reads as follows:

```
REPEAT
    Initialize the subdevices with states (at the next lower structural level)
    Simulate the superdevice at its next lower temporal level, if possible.
    Apply fault characteristics to narrow down the subdevice suspect-list
    Single step once through temporal hierarchy (if possible);
    Single step once through structural hierarchy;
    Eliminate candidates by asking for more information;
UNTIL fault diagnosed or basic level of structural and temporal hierarchy reached.
```

In the above algorithm, note that: (1) Representation scheme and control strategy are designed to be compatible.

(2) Stepping down the temporal hierarchy may not always be possible, because, a device may have only one level of temporal representation.

(3) Fault Characteristics are applied at the superdevice level, and at the superdevice's higher temporal level.

During candidate elimination, assuming complete visibility, measured values are asked at the ports of the candidates. These measured values are compared with expected values that are computed during the simulation stage of the device. Finally, the following algorithm is used to diagnose / eliminate candidates from further consideration:

```
For the device at hand:
IF output not as expected
    IF output cannot be explained by inputs
        IF device has subdevices
```

```
            generate suspects among subdevices
             and recurse on each of them.
        ELSE declare device to be faulty
    ELSE declare device to be correct
ELSE declare device to be correct
```

For example, Suppose a 4 bit sequential multiplier outputs 49 on inputs 6 and 8. The algorithm simulates the multiplier, at its next lower structural and temporal level, i.e., at subdevice level, for each clock cycle. Next, fault characteristics are applied on the superdevice, i.e., the multiplier to narrow down the list of subdevice suspects. Now, the algorithm steps down the structural hierarchy to the subdevice level (adder, 4 bit register, driver etc.) and down the temporal hierarchy to clock cycles. The user is asked for the values of the suspect subdevices during different clock cycles. These values are compared with the values computed during simulation earlier, to come up with a diagnosis.

A.6 Assumptions and their Relaxation

Assumptions are technical conveniences used to make diagnosis easier and faster at the expense of completeness. Some of the common assumptions made during diagnosis are :

(1) Single Fault Assumption
(2) Non-Canceling Fault Assumption
(3) Non-intermittent Fault Assumption

However, in real-life diagnosis, these assumptions are simplistic. They could potentially lead to failure of diagnosis and are hence undesirable. Therefore, assumptions are a matter of trade-off between efficiency and completeness. Ideally, the system VMES should carry on with the assumptions until it is infeasible to do so. This way, the system will have best of both worlds.

We outline a procedure below to relax single fault assumption when many sets of symptoms (input-output pairs) are available for diagnosis.

```
FOR every symptom DO
      find the suspect-list
      find the intersection of this list with
       that generated through previous symptoms
      IF intersection is null, relax single-fault assumption
END-FOR

IF assumption still holds, use only the intersection set
ELSE use the union set.
```

Assuming complete state visibility allows us to take two more liberties:

(1) We can relax non-intermittent fault assumption. The inputs and outputs of all subdevices are available during every clock cycle. Further, candidate elimination algorithm outlined earlier checks if measured output is justified by measured inputs. If this check yields false, irrespective of whether the fault is visible in other clock cycles, it is trapped.

(2) We can relax single fault assumption. Since all ports of all subdevices are measurable, faults can be contained and detected by measurement alone. When the measurements of a subdevice satisfy fault criteria of the candidate elimination algorithm, irrespective of whether other subdevices are faulty, the current subdevice can be declared faulty.

It should be noted that the assumption of complete visibility is simplistic. Ideally, the system VMES should be prepared to deal with situations where no values can be measured off some device. It should be able to work with measured values from only a few of the requested clock cycles. One solution would be to embed information about accessibility of ports and devices so that the system steers the diagnosis towards asking only those values that can be measured. Another would be to make the system flexible so that it can work with partial / alternate data. These ideas have not yet been incorporated into the current system.

A.7 Conclusion

We have proposed an algorithm to represent complex sequential circuits so as to facilitate diagnosis. Having outlined candidate generation based on electrical behavior, we have extended it to introduce structural information into sequential device diagnosis. With the proposed control structure of structural and temporal single-stepping, we have attempted to exploit the advantages of complete state visibility. Finally, we have produced an algorithm to relax single-fault assumption towards making diagnosis efficient and correct.

B A Scheme for Shadowing General Knowledge by Its Instances

This section describes new schemes for general AI reasoning systems focusing on the improvement of reasoning performance.

B.1 Introduction

The performance of an expert reasoning system is mainly dependent upon how it represents knowledge and how it controls reasoning. Control is needed to resolve knowledge conflicts in which more than one rule is applicable in some problem solving situation. Most expert systems tend to prefer rules of more specific features over rules of more general features as a way of conflict resolution [Sauers, 1988]. This brings the issue of `generality in knowledge` asking how the system distinguishes between more general and less general knowledge? In rule-based systems, the level of generality or specificity of a rule is determined by recognizing `special case` relationship between rules. A relative specificity between rules is defined by McDermott and Forgy [McDermott and Forgy, 1978] : a rule `r1` is more specific than another rule `r2` if (1) the two rules are not equal, (2) `r1` has at least as many antecedent clauses as `r2`, and (3) for each antecedent clause in `r2`, with constant elements $C_1, \cdots, C_n$, there exists a corresponding antecedent in `r1`, with constant elements $C'_1, \cdots, C'_m$, such that $\{C_1, \cdots, C_n\}$ is a subset of $\{C'_1, \cdots, C'_m\}$. According to this definition, a rule $A(a,b) \ \& \ B(a,b) \Rightarrow C(a,b)$ is treated as more specific than other rules like $A(a,b) \Rightarrow C(a,b)$, $\forall x \ \{A(a,x) \ \& \ B(a,x) \Rightarrow C(a,x)\}$, or $\forall x,y \ \{A(x,y) \ \& \ B(x,y) \Rightarrow C(x,y)\}$. This definition helps us to capture some abstract sense of what is meant by `more general` or `less general` in knowledge.

The concept of knowledge generality may be extended toward the depth of knowledge. In other words, we now intend to classify knowledge in terms of how it qualitatively contributes to solve problems. Some knowledge has the form of `Observation` $\Rightarrow$ `Conclusion` which directly associates inputs with some actions, but does not necessarily provide a reason for the relation between a pair [Chandrasekaran and Mittal, 1983]. We refer this kind of knowledge as `shallow knowledge`. In general, shallow knowledge has no underlying representation of causality or basic physical principles [Hart, 1982], instead it is just a collection of heuristic information such as statistical intuition or past experience of human experts [Reiter, 1987]. Shallow knowledge is usually represented by IF-THEN-like production rules. A typical system which uses shallow knowledge is MYCIN [Shortliffe, 1976]. MYCIN's knowledge base contains a collection of rules describing the relationships between symptoms and disease hypotheses, without specifying the causal links between them [Sembugamoorthy and Chandrasekaran, 1986].

For instance, MYCIN's `steroid rule` [Clancey, 1983] is represented as :

```
IF  (1) the infection which requires therapy is meningitis,
    (2) only circumstantial evidence is available for this case,
    (3) the type of the infection is bacterial,
    (4) the patient is receiving corticosteroids,
THEN  there is evidence that the organism which might be causing
      the infection are e.coli (.4), klebsiella-pneumoniae (.2),
      or pseudomonas-aeruginosa (.1)
```

On the other hand, `deep knowledge` contains lower-level, causal, and functional information using a qualitative model of the system [Yoon and Hammer, 1988]. There seems to be no strict form for deep knowledge structure, but several alternatives of representing deep knowledge can be summarized in [Chandrasekaran and Mittal, 1983] as : mathematical and simulation models, fundamental physical laws, functional and structural models of a device, causal networks, and sequences of cause effect rules which deduces consequences of events. In medical applications, CASNET [Weiss *et al.*, 1978] is an example system that is based on causal network. A CASNET model consists of observations of a patient and disease categories, which are also components of MYCIN, but it also maintains pathophysiological states that are associated with observations. These states form a network of cause-effect relationships, and patterns of states in the network are related to individual disease classifications. CASNET can explain more deeply the basis on which the final decisions are made about possible diseases.

So far we have discussed various descriptions about the general and specific knowledge distinction and also the deep and shallow knowledge distinction. While the former deals with only the syntactic features

by concentrating on the format of the knowledge, the latter is a rather semantical interpretation with vast number of model-theoretic definitions. We will mainly consider the general and specific knowledge distinction since it is easily recognized by the system, but we also expect some of the deep and shallow knowledge representations can be handled with a little modification.

It has been claimed that systems with deep or general knowledge can solve problems of greater complexity than systems with specific knowledge can [Hart, 1982]. But that is not the only requirement for any expert system. We sometimes give more priority to the goodness of the answer, or the reasonable cost to get the answer, rather than the broadness of solving power [Feigenbaum *et al.*, 1971]. It is widely admitted that reasoning by specific knowledge causes less system overload since several intermediate steps are omitted. As a result, specific knowledge will significantly contribute to the good performance of the system. While specificity is needed in a viewpoint of an expert system developer, generality is also needed for a problem-solving researcher.

We propose a systematic way to satisfy both requirements by recognizing generality relations in knowledge. Our approach is divided into 3 different issues : (1) Construction of a multi-level knowledge base by integrating different kinds of knowledge at different levels of generality, (2) Automatic migration of specific knowledge from general knowledge during a reasoning process, and (3) Shadowing general knowledge to select the most specific knowledge when several candidates are applicable.

A multi-level knowledge model is suggested to get benefits from both general knowledge and domain-specific shallow knowledge. It is intended to give the system the power of generalizability as well as good performance. Mostly deep or general knowledge is domain-independent, which implies that solving a problem by deep knowledge will need some additional steps of inference. For expert systems of real domains that require a large number of rule activations, we can expect that domain independent knowledge leads to serious performance degradation. Our goal is to use domain specific knowledge as far as possible, but the problem is how to relate the general knowledge to its specific counterpart.

B.2 Automatic Migration of General to Specific Knowledge

A motivation for the idea of migration comes from the observation of the knowledge derivation mechanism in a deductive reasoning system. The main task of a deductive reasoning system is to derive implicit knowledge from existing knowledge which are known to be true. After derivation, deduced information will be asserted into the knowledge base. In addition to directly derivable knowledge, however, we may get extra information which could be useful for the future reasoning.

To explain this, consider as an example a rule describing the characteristics of a transitive relation between two objects.

Rule1 : ∀R {transitive(R) ⇒ ∀x,y,z {R(x,y) & R(y,z) ⇒ R(x,z)}}

Rule1 reads: For any relation R which has the property of transitivity, if the relation R holds between x and y, and holds between y and z, then the relation R also holds between x and z. In order to show how the reasoning system automatically deduces new facts, consider a knowledge base contains the following facts as well as Rule1.

Fact1 : transitive(supports).
Fact2 : supports(a,b).
Fact3 : supports(b,c).
Fact4 : supports(c,d).

Now we want to infer supports(a,c) from this knowledge base. A natural deduction derivation [Bibel, 1986] could generate a sequence of inferencing to make supports(a,c) true :

Prop1 : {transitive(supports) ⇒
∀x,y,z {supports(x,y) & supports(y,z) ⇒ supports(x,z)}}
from Rule1 by Universal Instantiation
with a binding {supports/R}[1]

[1] A binding has a form of {term1/var1, term2/var2, ··· }

$\Downarrow$

Prop2 : ∀x,y,z {supports(x,y) & supports(y,z) ⇒ supports(x,z)}
from Prop1 and Fact1 by Modus Ponens

$\Downarrow$

Prop3 : {supports(a,b) & supports(b,c) ⇒ supports(a,c)}
from Prop2 by Universal Instantiation
with a binding {a/x, b/y, c/z}

$\Downarrow$

Prop4 : {supports(a,c)}
from Prop3, Fact2, and Fact3 by AND-introduction
and Modus Ponens

Note that Prop2 is a useful rule to keep around, and we call it Rule2.

Rule2 : ∀x,y,z {supports(x,y) & supports(y,z) ⇒ supports(x,z)}

Although initially not requested, the derivation of Rule2 can be justified according to the cognitive aspect of human reasoning. This means that since the relation supports becomes known to be transitive in this reasoning, from now on any cognitive agent also should know the nature of transitive relationship for supports by Rule2. Rule2 is an instance of Rule1, and it is a more specific rule than Rule1 by the specificity definition introduced in Section B.1. So after the derivation we could assert Rule2 into the knowledge base as well as supports(a,c). This is an example of a migration of general to specific knowledge during the inference.

As shown by the format of Rule1, the concept of migration raises the importance of a scheme of representing a nested rule, or an embedded rule. The specificity level of a migrated rule will be determined by the form of a more general rule represented by nesting with some quantifiers. A semantic interpretation for the rule nesting might say that the embedded definition delivers the intention of a rule builder about the usage of the rule. In the aforementioned example, Rule2 is migrated during the derivation of supports(a,c) according to the form of Rule1 such that only R is universally quantified at the outmost level. The intention of this rule can be interpreted as finding transitive relationships between objects without having any particular objects in mind.

To explain this more, suppose Rule1 is represented with different quantifier declarations such as Rule1_a or Rule1_b :

Rule1_a : ∀R,x {transitive(R) ⇒ ∀y,z {R(x,y) & R(y,z) ⇒ R(x,z)}}
Rule1_b : ∀R,x,y {transitive(R) & R(x,y) ⇒ ∀z R(y,z) ⇒ R(x,z)}}

Rule2_a and Rule2_b might be migrated from Rule1_a and Rule1_b, respectively, in the derivation of supports(a,c).

Rule2_a : ∀y,z {supports(a,y) & supports(y,z) ⇒ supports(a,z)}
Rule2_b : ∀z {supports(b,z) ⇒ supports(a,z)}

These rules may also be useful for some particular applications in which the rule builder has some specific objects in mind, or the knowledge base has many entries about the relationship between particular objects. In this example, Rule2_a focuses on the object a, and Rule2_b on a and b. Note that Rule2, Rule2_a, and Rule2_b have different levels of generality. This implies that different rules at different levels of generality can be migrated from the same kind of rule with just different declarations of universal quantifiers.

The mechanism of representing rule nesting is not emphasized in most automated reasoning systems, especially those systems based on resolution and unification. Although the definition of a well-formed formula in the resolution-based system [Robinson, 1965] allows embedded representations, those rules need to be translated into clause form in order to apply the resolution strategy. During this process of translating embedded representations into a flat structure such as clause form, the system loses the information about the rule nesting.

For instance, in a resolution-based system, Rule1 is translated into clause form by a sequence of transformations :

Fact1 : `transitive(supports)`
Fact2 : `supports(a,b)`
Fact3 : `supports(b,c)`
Fact4 : `supports(c,d)`
Fact5 : `supports(a,c)`
Rule1 : $\forall$R {transitive(R) $\Rightarrow$ $\forall$x,y,z {R(x,y) & R(y,z) $\Rightarrow$ R(x,z)}}
Rule2 : $\forall$x,y,z {supports(x,y) & supports(y,z) $\Rightarrow$ supports(x,z)}

Figure 19: A knowledge base after the derivation of supports(a,c)

$\neg$ transitive(R) $\vee$ $\neg$ holds(R,x,y) $\vee$ $\neg$ holds(R,y,z) $\vee$ holds(R,x,z)[2]

There is no difference among Rule1, Rule1_a, and Rule1_b in this system since all three rules are uniformly transformed to the same clause form. Some specific rules might be migrated from this kind of system, but it has no ability to recognize which one is useful in a particular reasoning. This observation leads to our claim that any system which intends to realize the concept of migration is supposed to have a method of utilizing the characteristic of embedded representations with quantifiers.

B.3 Shadowing General Knowledge by Its Instances

This section proposes a method of recognizing general and specific knowledge from a collection of knowledge at various levels of generality. A scheme of shadowing is suggested to select the most specific knowledge whenever several candidates are waiting for rule activation. This is important to accomplish our goal of performance enhancement, and it also makes it possible to apply only domain dependent rules as far as we can in the expert system applications.

A motivation for the idea of shadowing is illustrated by reconsidering the transitive relation example. In Section B.2, a natural deduction derivation is made to infer supports(a,c) from a given knowledge base. After the derivation, the system experiences a knowledge augmentation by asserting supports(a,c) and Rule2 into the knowledge base. The expanded knowledge base is shown in Fig 19.

Now we want to derive another implicit fact supports(b,d) from this knowledge base. A natural deduction for this problem is expected to be divided into two branches, where one branch of the derivation starts from Rule1 just like the previous derivation of supports(a,c), and the other branch considers Rule2 first. A sequence of the deduction in the first branch is described as :

Prop5 : {transitive(supports) $\Rightarrow$
$\forall$x,y,z {supports(x,y) & supports(y,z) $\Rightarrow$ supports(x,z)}}
from Rule1 by Universal Instantiation
with a binding {supports/R}
$\Downarrow$
Prop6 : $\forall$x,y,z {supports(x,y) & supports(y,z) $\Rightarrow$ supports(x,z)}
from Prop5 and Fact1 by Modus Ponens
$\Downarrow$
Prop7 : {supports(b,c) & supports(c,d) $\Rightarrow$ supports(b,d)}
from Prop6 by Universal Instantiation
with a binding {b/x, c/y, d/z}
$\Downarrow$
Prop8 : {supports(b,d)}
from Prop7, Fact3, and Fact4 by AND-introduction
and Modus Ponens

The second deduction branch is described as :

Prop9 : {supports(b,c) & supports(c,d) $\Rightarrow$ supports(b,d)}
from Rule2 by Universal Instantiation
with a binding {b/x, c/y, d/z}

[2] holds predicate is introduced to be consistent with a first-order logic

$$\Downarrow$$

Prop10 : {`supports(b,d)`}
from Prop9, Fact3, and Fact4 by AND-introduction
and Modus Ponens

Note that these two branches form a OR-branch such that **`supports(b,d)`** can be derived by either one of two branches.

Several important points are worth mentioning from this observation. First of all, there are some duplicate reasoning steps in the first branch using **`Rule1`**, compared with the derivation of **`supports(a,c)`**. The similarity in the reasoning steps between these two derivations is expected since two reasonings share the same constant **`supports`**, which means they are in the same specific domain of **`supports`**. Another notable phenomenon is in the second branch. The reasoning steps in this branch are reduced to 2 for the derivation of **`supports(b,d)`**, compared with 4 steps in the previous inference of **`supports(a,c)`** and also in the first branch of **`supports(b,d)`**. Nothing goes better if we should be able to activate only this second branch with cutting off the first branch. This is the main objective of the shadowing scheme.

The issue now is how to systematically relate more general knowledge to its instances. In order to take advantage of the previously acquired information, we need a way of memorizing instances with respect to the corresponding general knowledge. At this point, a method of saving the instances looks important. What we suggest to maintain is a list of instance information for each rule. Each instance element in the list has at least two kinds of information : the identification or the name of the instance, and a binding information explaining how the instance is related to the rule.

More formally, an instance list for a rule **G** has the form of

$$((\mathbf{S}_1, \boldsymbol{\sigma}_1), (\mathbf{S}_2, \boldsymbol{\sigma}_2), \cdots, (\mathbf{S}_n, \boldsymbol{\sigma}_n)),$$

where n is the number of known instances of **G**, $\mathbf{S}_i$ is the name of the ith instance of **G**, and $\boldsymbol{\sigma}_i$ represents a binding which unifies **G** and $\mathbf{S}_i$. In fact, $\mathbf{S}_i$ can be either a name, or a pointer to the actual structure of the ith instance depending on particular implementations. All lists are initially empty, and they are dynamically updated as the inference goes on. In the transitive relation example, `Rule1` will be attached with an instance list after deriving `Prop2` such as `((Rule2, {supports/R}))`.

Once we defined the method of storing instances, the next step is to formulate a way of how to use them. Suppose a rule **P** has a list of known instances $((\mathbf{S}_1, \boldsymbol{\sigma}_1), (\mathbf{S}_2, \boldsymbol{\sigma}_2), \cdots, (\mathbf{S}_n, \boldsymbol{\sigma}_n))$ as defined above. Each binding in a known instance list $\boldsymbol{\sigma}_i$ contains only free variable substitutions. Also assume **fv-list** = { $\boldsymbol{fv}_1, \boldsymbol{fv}_2, \cdots, \boldsymbol{fv}_m$ } is a list of free variables of **P**. If, at some stage in a derivation, a proposition is deduced from **P** by universal instantiation with a binding $\boldsymbol{\phi}$, we check the information in $\boldsymbol{\phi}$ with each $\boldsymbol{\sigma}_i$ $(1 \leq i \leq n)$ before any further action is made. We can stop this branch of derivation if the condition for shadowing is satisfied :

The rule **P** will be shadowed from the inference in the case that for any one of $i(1 \leq i \leq n)$, each substituted value for a free variable in $\boldsymbol{\sigma}_i$ is the same as the value for the same variable in $\boldsymbol{\phi}$.

Determining whether a variable in $\boldsymbol{\phi}$ is free is done by checking the list membership for **fv-list**. For instance, consider the first deduction branch of `supports(b,d)`. When **`Prop5`** is being made from **`Rule1`**, we have a situation in which

$\boldsymbol{\phi}$ = {`supports/R`}
$\boldsymbol{\sigma}$ = {`supports/R`}
fv-list = {`R`}.

Since `R` is the only free variable of **`Rule1`**, and the bound values for `R` in $\boldsymbol{\phi}$ and $\boldsymbol{\sigma}$ are the same, the shadowing condition is satisfied. At this point, we have enough evidence that there is a more specific rule solely capable of deriving the given fact more efficiently. Therefore, the first branch is blocked here and will not be proceeded any further.

The evaluation of the shadowing may be made in two ways. Firstly, we can anticipate the shadowing makes the inference of `supports(b,d)` faster than in a non-shadowing normal inference because it is clear that there is some reductions of reasoning steps with the help of shadowing. Secondly, we hope that the

improvement goes further so that the inference of **supports(b,d)** is even faster than the first inference of supports(a,c) if the shadowing is adopted. Consequently, the system is now equipped with some intelligence which automatically prunes some inference branches using the previous reasoning experience.

B.4 An Implementation : SNePS

SNePS is a knowledge representation/ reasoning system using an intentional semantic network formalism [Shapiro, 1979]. SNePS is equipped with the ability to represent nested rules in any levels of depth. The inference package of SNePS (SNIP) provides an object-orient style of reasoning with assigning a process to each network node and maintaining several registers to keep information necessary for the message passing between processes. These registers are useful to save the instance information for shadowing.

Some peculiar features embodied in SNIP [Hull, 1986] are described in detail.

- SNIP treats inference as an activation of the network itself, rather than a compilation of the network into a distinct active connection graph of processes. (The latter method was adopted in old version of SNIP [Shapiro, 1977])
- There is a smaller set of processes and the types of processes are limited to the types of nodes found in the network. Current version of SNIP has 3 types of processes, that is **a proposition node process**, **a rule node process**, and **a user process**.
- Node processes are directly attached to the network nodes and the communications are made through **channels** which are incoming and outgoing paths between processes.

There are two types of messages which will be sent between node processes, **reports** and **requests.** The **reports** message contains substitutions which represent instances which are known to be true. The **requests** message contains desired substitutions, and the necessary information to set up the channels through which reports of these instances can be sent.

Each node process has a set of registers which actually set up the channel for message passing. Processes send and receive messages, and perform inferences based only on these messages and the register information. Some of the registers are described here.

- **known-instances** - the collection of instances of this node which are known to be true (both positive and negative)
- **reports** - the collection of reports received from other node processes
- **requests** - the collection of requests received from other node processes
- **incoming-channels** - the set of channels which will be feeding instance reports to this node
- **outgoing-channels** - the set of channels to which this node is to report instances that are discovered

A view of a message passing between two processes is shown in Figure 20.

To illustrate how SNIP realize the mechanism of the migration and the shadowing, we visit the transitive rule example again. Figure 21 (a) shows SNePSUL (SNePS User Language) format to represent the knowledge used in the example, and SNePS internal representation for these rules are described in Figure 21 (b). Here M represents a molecular node, and ! symbol indicates that the node is asserted at the top level. P and V denote a pattern node and a variable node, respectively [Shapiro, 1979].

Actual reasoning for **supports(a,c)** is done by **deduce** command as shown in Figure 21 (a), which builds an unasserted node **M6** and initiates a backward chaining to derive M6 from the given knowledge base.

SNIP assigns a process to each SNePS node when it is involved during the inference. The inference is performed solely by message passing between processes via channels. Channels between two processes are created if their corresponding nodes are pattern matched or, in case of rules, one node is unifiable with the consequent of the other node. Those channels made by rules are used for implementing rule chaining. The deduction of **supports(a,c)** in SNIP proceeds with the following steps.

Initially a user process is created and invokes process **M6**[3]. **M6** sends a request to **P4** by setting the **requests** register of **P4** to a substitution {**supports/V1, a/V2, c/V4**}. Since **P4** is the consequent of **P5**,

[3] Processes are named after their corresponding SNePS node names.

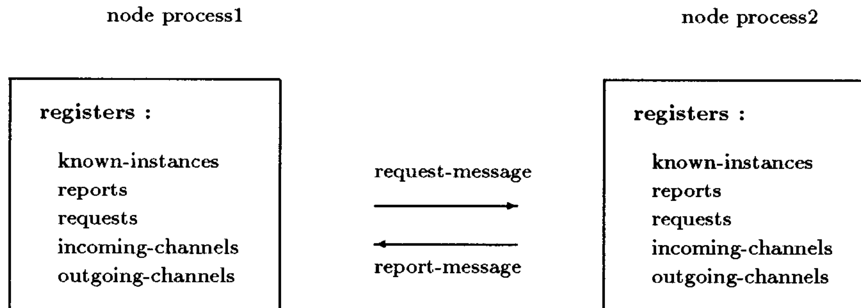


Figure 20: A message passing between SNIP processes

```
(assert forall $r
        ant (build member *r class transitive)
        cq (build forall ($x $y $z)
                 &ant ((build agent *x act *r object *y)
                       (build agent *y act *r object *z))
                 cq (build agent *x act *r object *z)))

(assert agent a act supports object b)
(assert agent b act supports object c)
(assert agent c act supports object d)
(assert member supports class transitive)
(deduce agent a act supports object c)
```

(a)

```
(M1! (FORALL V1)
     (ANT (P1 (MEMBER V1) (CLASS TRANSITIVE)))
     (CQ (P5 (FORALL V2 V3 V4)
             (&ANT (P2 (AGENT V2) (ACT V1) (OBJECT V3))
                   (P3 (AGENT V3) (ACT V1) (OBJECT V4)))
             (CQ   (P4 (AGENT V2) (ACT V1) (OBJECT V4))))))

(M2! (AGENT A) (ACT SUPPORTS) (OBJECT B))
(M3! (AGENT B) (ACT SUPPORTS) (OBJECT C))
(M4! (AGENT C) (ACT SUPPORTS) (OBJECT D))
(M5! (MEMBER SUPPORTS) (CLASS TRANSITIVE))
(M6 (AGENT A) (ACT SUPPORTS) (OBJECT C))
```

(b)

Figure 21: SNePS representations for the transitive relation example

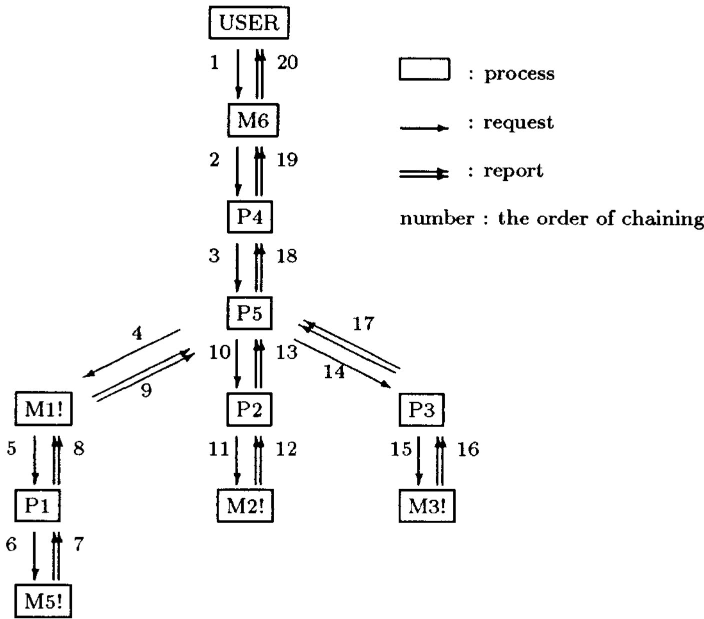


Figure 22: Process activation graph for **supports(a,c)**

P4 sends the request to P5 with {supports/V1, a/V2, c/V4}. P5 is a rule node, but no known instances exist. So P5 sends the request to its dominating rule node M1! with {supports/V1}, since V1 is the only free variable in P5. M1! is an asserted rule node. So M1! sends the request to its antecedent P1 with {supports/V1}. P1 is matched with asserted proposition node M5!. P1 sends a report back to M1! by setting the **reports** register of M1! to M5!, which then is sent back to P5. Now the free variable of P5 is bound to supports. A migration takes place at this point. A new rule M7!, which is an instance of P5, is now asserted and the **known-instances** register of P5 is set to M7!. The SNePS representation of the newly instantiated rule M7! is shown below.

```
(M7! (FORALL V2 V3 V4)
     (&ANT (P6 (AGENT V2) (ACT SUPPORTS) (OBJECT V3))
           (P7 (AGENT V3) (ACT SUPPORTS) (OBJECT V4)))
     (CQ (P8 (AGENT V2) (ACT SUPPORTS) (OBJECT V4))))
```

The exact content of the **known-instances** register of P5 will be :

```
*KNOWN-INSTANCES* = ((((P5 . M7!) (V1 . SUPPORTS)) . SNIP::POS))
```

This says M7! is a positive instance of P5 with the binding of {supports/V1}. Notice that the **known-instances** register has not only the name of the instance, but also the binding information for free variables, which is necessary to verify the appropriate instances. After migration, the inference goes on. P5 now sends a request to its antecedents, P2 and P3. Since P2 and P3 are matched with M2! and M3!, respectively, reports are sent from P2 and P3 to P5. The report from P5 is sent back to P4, then to M6, and then to the user process. Finally M6 is asserted as M6!. This process activation with messages passing through channels is drawn in Figure 22.

Now suppose we want to infer supports(b,d). SNePS builds M8 for this node and starts a deduction.

```
(M8 (AGENT B) (ACT SUPPORTS) (OBJECT D))
```

The inference procedure becomes more complicated because the knowledge base now has a migrated rule M7! as well as M1!.

Initially a user process is created and invokes process M8. M8 is matched with both P4 and P8. So now the inference goes with two branches. While a request is sent from M8 to P4 with a substitution {supports/V1,

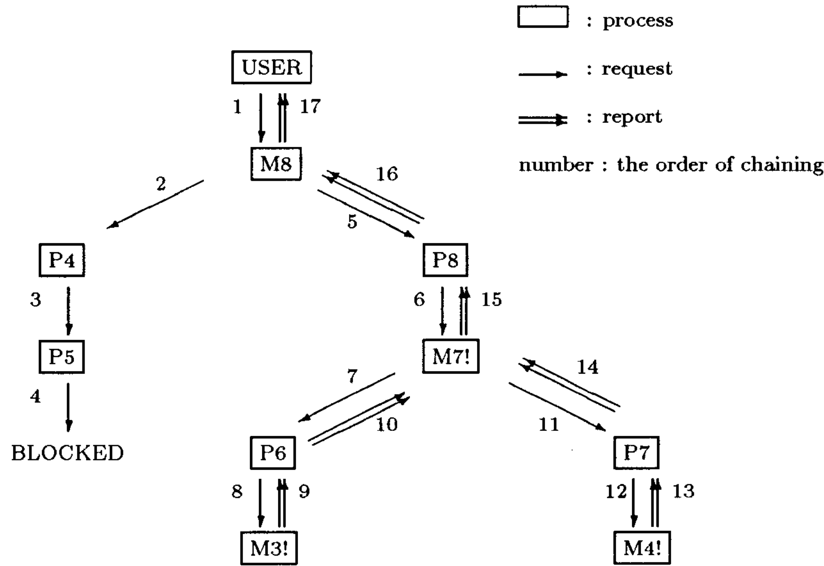


Figure 23: Process activation graph for **supports(b,d)**

unit : seconds

	without shadowing	with shadowing
`supports(a,c)`	6.85	5.92
`supports(b,d)`	10.37	3.85

Table 1: Execution time comparisons for the transitive relation rule

`b/V2, d/V4`}, `M8` sends another request to `P8` with a different substitution {`b/V2, d/V4`}. Note that there is no substitution for `V1` because `P8` has no such variable. Since `P4` is the consequent of `P5`, `P4` sends the request to process `P5` with {`supports/V1, b/V2, d/V4`}. `P8` is also the consequent of `M7!`, so `P8` sends the request to `M7!` with {`b/V2, d/V4`}. These steps of requests sending can proceed in parallel.

When process `P4` is sending a request to `P5` with the substitution of {`supports/V1, b/V2, d/V4`}, the **requests** register of `P5` is set to :

```
*REQUESTS* = ((((P4 . M8) (V1 . SUPPORTS) (V4 . D) (V2 . B))
              NIL  P4  OPEN))
```

This structure tells that a request is sent from `P4` via an open channel, and the requested substitution is {`supports/V1, b/V2, d/V4`}. Now we check the shadowing condition mentioned in Section B.3. Since the process `P5` has `M7!` as an instance in the **known-instances** register, the next step is to verify that `M7!` is a useful instance in this particular inference. The filtering process compares the binding information for free variables in both registers. Eventually `M7!` is accepted as a proper instance because the substitution for free variable `V1` of `P5` in the **requests** register is identical to that in the **known-instances** register. Finally the activation of `P5` is blocked and the process `M7!` will be responsible for the remaining inference. Process activation graph for this part is drawn in Figure 23.

We ran this example by SNIP on TI Explorer, and Table 1 shows the time comparisons for the execution of these rules. We compare the time between `supports(a,c)` and `supports(b,d)`, and also between the case that the shadowing is implemented and the case without shadowing. The execution of `supports(b,d)` is done after the more specific rule is generated by `supports(a,c)`.

In this table, we will see the reduced time for `supports(b,d)` from 10.37 to 3.85 by shadowing. Furthermore, we can also notice from the column named `with shadowing` that the inference of `supports(b,d)`

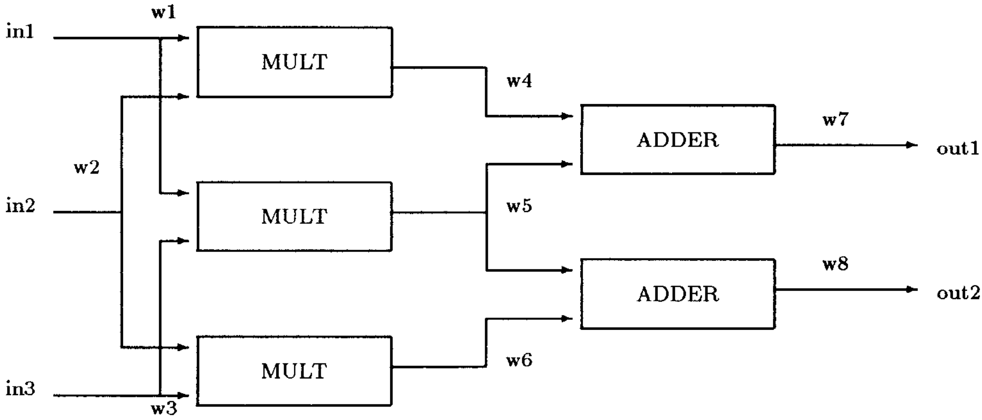


Figure 24: A logical abstraction for M3A2

is even faster between `supports(a,c)`. This result tells the real performance enhancement obtained by applying the scheme of shadowing.

B.5 An Application

Some diagnostic expert systems use a model of devices which is structural or functional [Davis, 1984; Taie, 1987]. This approach has been used to find a faulty component in a digital combinational circuit like M3A2. M3A2 is a simple circuit which has 3 multipliers and 2 adders as shown in Figure 24.

The values of outputs are determined by inputs as :

```
out1 = in1 * in2 + in1 * in3
out2 = in1 * in3 + in2 * in3
```

If the calculated values of outputs from given inputs are different from the measured values, a violation is detected and the diagnosis starts to locate the faulty components. The component could be a device or a wire, so we need diagnostic rules for such components.

An example we will show is a wire faulty detection for M3A2. There are several types of wires used in this circuit analysis. For instance, WIRE3 denotes a type of wires connected to three different components, and WIRE2 denotes a different type of wires connected to two different components. In Figure 24, **w4**, **w6**, **w7**, and **w8** fall under the category of WIRE2, and **w1**, **w2**, **w3**, and **w5** have WIRE3 property. We can build a diagnose rule for detecting wire faults which generally applies to different types of wires.

$$\forall \text{T} \ \{ \ \text{Wire-type(T)} \Rightarrow \\ \forall \text{O,P1,P2} \ \{ \ \text{T(O)} \ \& \ \text{Bi-port(O,P1,P2)} \ \& \ \text{value(P1)} \neq \text{value(P2)} \\ \Rightarrow \text{faulty(O)} \ \}\}$$

A SNePS representation for this rule is shown in Figure 25.

Suppose the first diagnose is for **w7** of WIRE2. After migration, a specific rule is generated for wires of type WIRE2 by replacing the free variable T of the general rule by WIRE2.

$$\forall \text{O,P1,P2} \ \{ \ \text{Wire2(O)} \ \& \ \text{Bi-port(O,P1,P2)} \ \& \ \text{value(P1)} \neq \text{value(P2)} \\ \Rightarrow \text{faulty(O)} \ \}$$

A SNePS representation for this migrated rule is shown in Figure 26.

This rule will shadow the original rule when another reasoning is performed for a wire of the same type **w8**. Table 2 show the improvement of performance by comparing two executions with or without shadowing.

Shadowing can be very effective especially for the applications which perform diagnosis about similar components many times. So far we illustrate the potential applicability of the migration and the shadowing

```
(M2! (FORALL V1)
    (ANT (P1 (TYPE V1) (TYPE-CLS  WIRE)))
    (CQ (P11  (FORALL V2 V3 V4)
              (&ANT (P2 (OBJECT V2) (TYPE V1))
                    (P3 (BI-PORT1 V2) (OBJECT V3))
                    (P4 (BI-PORT2 V2) (OBJECT V4)))
              (CQ (P10  (FORALL V5 V6)
                        (&ANT (P6 (OBJECT V3)
                                  (ATTR (P5  (ATRB V5)
                                             (ATRB-CLS  M-VALUE)
                                             (MODALITY LOGICAL))))
                              (P8 (OBJECT V4)
                                  (ATTR (P7  (ATRB V6)
                                             (ATRB-CLS  M-VALUE)
                                             (MODALITY LOGICAL)))))
                        (CQ (P9 (OBJECT V2) (TYPE V1)
                                (ATTR (M1  (ATRB FAULTY)
                                           (ATRB-CLS  STATE)
                                           (MODALITY LOGICAL)))))))))
```

Figure 25: SNePS representation for a wire-faulty detection rule

```
(M10! (FORALL V2 V3 V4)
      (&ANT (P2 (OBJECT V2) (TYPE WIRE2))
            (P3 (BI-PORT1 V2) (OBJECT V3))
            (P4 (BI-PORT2 V2) (OBJECT V4)))
      (CQ (P10  (FORALL V5 V6)
                (&ANT (P6 (OBJECT V3)
                          (ATTR (P5  (ATRB V5)
                                     (ATRB-CLS  M-VALUE)
                                     (MODALITY LOGICAL))))
                      (P8 (OBJECT V4)
                          (ATTR (P7  (ATRB V6)
                                     (ATRB-CLS  M-VALUE)
                                     (MODALITY LOGICAL)))))
                (CQ (P9 (OBJECT V2) (TYPE WIRE2)
                        (ATTR (M1  (ATRB FAULTY)
                                   (ATRB-CLS  STATE)
                                   (MODALITY LOGICAL))))))))
```

Figure 26: SNePS representation for a migrated wire-faulty rule

unit : seconds

	without shadowing	with shadowing
faulty(w7)	39.18	38.93
faulty(w8)	43.22	27.10

Table 2: Execution time comparisons for wire faulty detection rule

scheme to the real domain of applications. We would like to explain in the next section the details about the implementation.

B.6 Conclusion

An automatic scheme is suggested for migrating specific knowledge and for shadowing deep knowledge by its instances in an expert reasoning system. The motivation of this work is to make the inference faster in a multi-level knowledge system in which various kinds of knowledge are present. Migration and shadowing schemes enable the system to accomplish the performance improvement as well as the system generalization. Most specific knowledge is preferred to be selected among candidates at each stage of the inference. Experimental results have shown that the inference speed is significantly improved with shadowing method. Applicability to real complex domain should be further tested.

References

[Bibel, 1986] Wolfgang Bibel. Methods of automated reasoning. In W. Bibel and Ph. Jorrand, editors, *Fundamentals of Artificial Intelligence: An Advanced Course*, pages 171–217. Springer-Verlag, 1986.

[Bond, Jr. and Rigney, 1966] N. A. Bond, Jr. and J. W. Rigney. Bayesian aspects of trouble shooting behavior. *Human Factors*, 8:377–383, October 1966.

[Buchanan and Shortliffe, 1984] B. G. Buchanan and E. H. Shortliffe, editors. *Rule-based Expert Systems*. Addison-Wesley, Reading, MA, 1984.

[Chandrasekaran and Mittal, 1983] B. Chandrasekaran and S. Mittal. Deep versus compiled knowledge approaches to diagnostic problem-solving. *International Journal of Man-Machine Studies*, 19:425–436, 1983. Also in *Developments in Expert Systems*, pages 23–34. Academic Press, London, 1984.

[Chen and Srihari, 1989] J. Chen and S. N. Srihari. Candidate ordering and elimination in model-based fault diagnosis. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1363–1368. Morgan Kaufmann, 1989.

[Clancey, 1983] W. J. Clancey. The epistemology of a rule-based expert system: A framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[Coppola, 1984] A. Coppola. Artificial intelligence applications to maintenance. In *Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance*, pages 23–43, June 1984.

[Dale, 1957] H. C. A. Dale. Fault-finding in electronic equipment. *Ergonomics*, 1:356–385, 1957.

[Davis and Hamscher, 1988] R. Davis and W. Hamscher. Model-based reasoning: Troubleshooting. In H. Shrobe, editor, *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, chapter 8, pages 297–346. Morgan Kaufmann, San Mateo, CA, 1988.

[Davis and Shrobe, 1983] R. Davis and H. Shrobe. Representing structure and behavior of digital hardware. *Computer*, 16(10):75–82, October 1983.

[Davis *et al.*, 1982] R. Davis, H. Shrobe, W. Hamscher, K. Wieckert, M. Shirley, and S. Polit. Diagnosis based on description of structure and function. In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 137–142. Morgan Kaufmann, 1982.

[Davis, 1983] R. Davis. Diagnosis via causal reasoning: Paths of interaction and the locality principle. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 88–94, 1983.

[Davis, 1984] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24(3):347–410, 1984.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[Feigenbaum *et al.*, 1971] E. A. Feigenbaum, B. G. Buchanan, and J. Lederberg. On generality and problem solving: A case study using the DENDRAL program. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*, pages 165–190. Elsevier, New York, 1971.

[Geller *et al.*, 1987] J. Geller, M. R. Taie, S. C. Shapiro, and S. N. Srihari. Device representation and graphics interfaces of VMES. In *Proceedings of Applications of Artificial Intelligence in Engineering*, 1987.

[Geller, 1988] J. Geller. *A Knowledge Representation Theory for Natural Language Graphics*. PhD thesis, Department of Computer Science, SUNY at Buffalo, 1988. Technical Report 88-15.

[Genesereth, 1982] M. R. Genesereth. Diagnosis using hierarchical design models. In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 278–283, 1982.

[Genesereth, 1984] M. R. Genesereth. The use of design description in automated diagnosis. *Artificial Intelligence*, 24(3):411–436, 1984.

[Hamscher and Davis, 1984] W. Hamscher and R. Davis. Diagnosing circuits with state: An inherently underconstrained problem. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 142–147. Morgan Kaufmann, 1984.

[Hart, 1982] Peter E. Hart. Directions for AI in the eighties. *SIGART Newsletter*, 79:11–16, January 1982.

[Hartley, 1984] R. T. Hartley. CRIB: Computer fault-finding through knowledge engineering. *Computer*, pages 76–83, March 1984.

[Hayes-Roth *et al.*, 1983] F. Hayes-Roth, D. A. Waterman, and D. B. Lenant, editors. *Building Expert System.* Addison-Wesley, 1983.

[Hull, 1986] Richard G. Hull. A new design for SNIP the SNePS inference package. SNeRG Technical Note 14, Dept. of Computer Science, State University of New York at Buffalo, 1986.

[Hunt and Rouse, 1984] R. M. Hunt and W. B. Rouse. A fuzzy rule-based model of human problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(1):112–120, January/February 1984.

[McDermott and Forgy, 1978] J. M. McDermott and C. Forgy. Production system conflict resolution strategies. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern Directed Inference Systems.* Academic Press, New York, 1978.

[McKay and Shapiro, 1981] D. P. McKay and S. C. Shapiro. Using active connection graphs for reasoning with recursive rules. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 368–374, Los Altos, CA, 1981. Morgan Kaufmann.

[Michie, 1980] D. Michie. Expert systems. *The Computer Journal*, 23(4):369–376, 1980.

[Nelson, 1982] W. R. Nelson. REACTOR: An expert system for diagnosis and treatment of nuclear reactor accidents. In *Proceedings of the Second National Conference on Artificial Intelligence*, pages 296–301, 1982.

[Rasmussen and Jensen, 1974] J. Rasmussen and A. Jensen. Mental procedures in real-life tasks: A case study of electronic trouble shooting. *Ergonomics*, 17(3):293–307, 1974.

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

[Reggia *et al.*, 1985] J. A. Reggia, D. S. Nau, and P. Y. Wang. A formal model of diagnostic inference. I. Problem formulation and decomposition. *Information Sciences*, 37:227–256, 1985.

[Reiter, 1987] R. Reiter. A theory of diasgnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Robinson, 1965] J. A. Robinson. A machine oriented logic based on the resolution principle. *Journal of ACM*, 12:23–41, 1965.

[Rouse *et al.*, 1980] W. B. Rouse, S. H. Rouse, and S. J. Pellegrino. A rule-based model of human problem solving performance in fault diagnosis tasks. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(7):366–376, July 1980.

[Rouse, 1978a] W. B. Rouse. Human problem solving performance in a fault diagnosis task. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(4):258–271, April 1978.

[Rouse, 1978b] W. B. Rouse. A model of human decision making in a fault diagnosis task. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(5):357–361, May 1978.

[Rouse, 1979a] W. B. Rouse. Problem solving performance of first semester maintenance trainees in two fault diagnosis tasks. *Human Factors*, 21(5):611–618, October 1979.

[Rouse, 1979b] W. B. Rouse. Problem solving performance of maintenance trainees in a fault diagnosis task. *Human Factors*, 21(2):195–203, April 1979.

[Rouse, 1984] W. B. Rouse. Models of natural intelligence in fault diagnosis tasks: Implications for training and aiding of maintenance personnel. In *Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance*, pages 193–212, June 1984.

[Sauers, 1988] Ron Sauers. Controlling expert systems. In L. Bolc and M. J. Coombs, editors, *Expert System Applications*, pages 79–197. Springer-Verlag, 1988.

[Sembugamoorthy and Chandrasekaran, 1986] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. In J. Kolodner and C. Reisbeck, editors, *Experience, Memory, and Reasoning*, chapter 4, pages 47–73. Lawrence Eribaum Associates Publishers, 1986.

[Shapiro *et al.*, 1986] S. C. Shapiro, S. N. Srihari, M. R. Taie, and J. Geller. VMES: A network-based versatile maintenance expert system. In *Proceedings of the 1st International Conference on Applications of AI to Engineering Problems*, pages 925–936. Springer-Verlag, April 1986.

[Shapiro, 1977] Stuart C. Shapiro. Compiling deduction rules from a semantic network into a set of processes. In *Abstracts of Workshop on Automatic Deduction*, MIT, 1977.

[Shapiro, 1979] S. C. Shapiro. The SNePS semantic network processing system. In N. V. Findler, editor, *Associative Networks: The Representation and Use of Knowledge by Computers*, pages 179–203. Academic Press, 1979.

[Shapiro, 1982] S. C. Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *The American Journal of Computational Linguistics*, 8(1):12–15, 1982.

[Shapiro, 1986] S. C. Shapiro. Symmetric relations, intensional individuals, and variable binding. In *Proceedings of the IEEE 74*, pages 1354–1363, October 1986.

[Shortliffe, 1976] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN*. North-Holland, New York, 1976.

[Taie and Srihari, 1986] M. R. Taie and S. N. Srihari. Device modeling for fault diagnosis. In *Proceedings of the 2nd Expert Systems in Government Symposium*, pages 144–150, Washington, DC, October 1986. IEEE Computer Society Press.

[Taie and Srihari, 1987] M. R. Taie and S. N. Srihari. Modeling connections for circuit diagnosis. In *Proceedings of the Third Conference on Artificial Intelligence Applications*, pages 81–86, Orlando, FL, February 1987. IEEE Computer Society Press.

[Taie *et al.*, 1987] M. R. Taie, J. Geller, S. N. Srihari, and S. C. Shapiro. Knowledge based modeling of circuit boards. In *Proceedings of 1987 Annual Reliability and Maintainability Symposium*, pages 422–427, January 1987.

[Taie, 1987] M. R. Taie. *Representation of Device Knowledge for Versatile Fault Diagnosis*. PhD thesis, Department of Computer Science, SUNY at Buffalo, Buffalo, NY 14260, May 1987. Technical Report 88-07.

[Weiss *et al.*, 1978] S. M. Weiss, C. A. Kulikowski, S. Amarel, and A. Safir. A model-based method for computer-aided medical decision making. *Artificial Intelligence*, 11:145–172, 1978.

[Xiang and Srihari, 1985] Z. Xiang and S. N. Srihari. Spatial structure and function representation in diagnostic expert systems. In *Proceedings of the Fifth International Workshop on Expert Systems and Their Applications*, pages 191–206, Avignon, France, May 1985.

[Xiang and Srihari, 1986] Z. Xiang and S. N. Srihari. A strategy for diagnosis based on empirical and model knowledge. In *Proceedings of the Sixth International Workshop on Expert Systems and Their Applications*, pages 835–848, Avignon. France, April 1986.

[Yoon and Hammer, 1988] W. C. Yoon and J. M. Hammer. Deep-reasoning fault diagnosis: An aid and a model. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-18(4):659–676, 1988.