(4)



RADC-TR-89-259, Vol III (of twelve)
Interim Report
October 1989


# NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT - 1988 The Versatile Maintenance Expert System Research Project

Syracuse University

S.N. Srihari, S.C. Shapiro, S.J. Upadhyaya



*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

This effort was funded partially by the Laboratory Director's fund.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-89-259, Vol III (of twelve) has been reviewed and is approved for publication.

APPROVED: Dale W. Richards

DALE W. RICHARDS
Project Engineer

APPROVED: John J. Bart

JOHN J. BART
Technical Director
Directorate of Reliability & Compatibility

FOR THE COMMANDER: James W. Hyde III

JAMES W. HYDE III
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBES ) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific doucment require that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS<br>N/A |
| 2a. SECURITY CLASSIFICATION AUTHORITY<br>N/A | 3. DISTRIBUTION/AVAILABILITY OF REPORT<br>Approved for public release; distribution unlimited. |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE<br>N/A | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S)<br>N/A | 5. MONITORING ORGANIZATION REPORT NUMBER(S)<br>RADC-TR-89-259, Vol III (of twelve) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Northeast Artificial Intelligence Consortium (NAIC) | 6b. OFFICE SYMBOL (If applicable) |
| | 7a. NAME OF MONITORING ORGANIZATION<br>Rome Air Development Center (COES) |
| 6c. ADDRESS (City, State, and ZIP Code)<br>Science & Technology Center, Rm 2-296<br>111 College Place, Syracuse University<br>Syracuse NY 13244-4100 | 7b. ADDRESS (City, State, and ZIP Code)<br>Griffiss AFB NY 13441-5700 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION<br>Rome Air Development Center | 8b. OFFICE SYMBOL (If applicable)<br>COES |
| | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER<br>F30602-85-C-0008 |
| 8c. ADDRESS (City, State, and ZIP Code)<br>Griffiss AFB NY 13441-5700 | 10. SOURCE OF FUNDING NUMBERS |

| PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO | WORK UNIT ACCESSION NO |
|---|---|---|---|
| 62702F | 5581 | 27 | 13 |

11. TITLE (Include Security Classification)
NORTHEAST ARTIFICIAL INTELLIGENCE CONSORTIUM ANNUAL REPORT - 1988 The Versatile Maintenance Expert System Research Project

12. PERSONAL AUTHOR(S)
S. N. Srihari, S. C. Shapiro, S. J. Upadhyaya

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Interim | FROM Jan 88 TO Dec 88 | October 1989 | 56 |

16. SUPPLEMENTARY NOTATION This effort was funded partially by the Laboratory Directors' Fund. This effort was performed as a subcontract by the State University of New York at Buffalo to Syracuse University, Office of Sponsored Programs.

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Artificial Intelligence, Expert Systems, Reasoning, Fault Detection, User Interfaces, Graphical Knowledge, Semantic Networks, |
| 12 | 05 | | |
| 12 | 07 | | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Northeast Artificial Intelligence Consortium (NAIC) is sponsored by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose is to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress that has been made in the fourth year of the Consortium on the Versatile Maintenance Expert System research task. Technical areas addressed include candidate ordering, reordering and elimination, representation and diagnosis of sequential circuits, shadowing of deep knowledge and user interfaces. Work was initiated on the use of a more complex circuit as the testbed for this research.

| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS | UNCLASSIFIED |

| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE (Include Area Code) | 22c. OFFICE SYMBOL |
|---|---|---|
| Dale W. Richards | (315) 330-3476 | RADC (RBES) |

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Item 10. SOURCE OF FUNDING NUMBERS (Continued)

| Program Element Number | Project Number | Task Number | Work Unit Number |
|---|---|---|---|
| 62702F | 5581 | 27 | 23 |
| 61102F | 2304 | J5 | 01 |
| 61102F | 2304 | J5 | 15 |
| 33126F | 2155 | 02 | 10 |
| 61101F | LDFP | 27 | 01 |

# 3 THE VERSATILE MAINTENANCE EXPERT SYSTEM (VMES) RESEARCH PROJECT

Report submitted by:

Dr. Sargur N. Srihari, Principal Investigator
Dr. Stuart C. Shapiro, Co-Principal Investigator
Dr. Shambhu J. Upadhyaya, Co-Principal Investigator
Dr. James Geller, Research Associate
Jiah-shing Chen, Graduate Research Assistant
Amruth Kumar, Graduate Research Assistant
Joongmin Choi, Graduate Research Assistant
Sudip Nag, Graduate Research Assistant
Scott S. Campbell, Computer Programmer/Analyst

Department of Computer Science
SUNY at Buffalo
226 Bell Hall
Buffalo, NY 14260

TABLE OF CONTENTS

# 3 THE VERSATILE MAINTENANCE EXPERT SYSTEM (VMES) RESEARCH PROJECT

## 3.1 TECHNICAL OVERVIEW

### 3.1.1 Summary of 1988

Research on the Versatile Maintenance Expert System (VMES) project is concerned with issues in the development of a system to diagnose faults in an electronic circuit. The system reasons using a structural-functional model of the device. The model is built using a library of primitive components.

During 1988 our accomplishments were in the following areas: *(1)* a systematic method for initial candidate ordering which takes into account the structure of a device and the relationships of its components with both correct and incorrect outputs, *(2)* an interface for encoding devices in a previously developed method of device representation, *(3)* sequential circuit diagnosis, *(4)* the migration of deep knowledge to shallow knowledge, *(5)* natural language graphics, *(6)* continuing development of the new revision of the Semantic Network Processing System (SNePS-2). In the remainder of this summary, a few more details in each are given.

A major step in model-based fault diagnosis is the generation of candidate submodules which might be responsible for the observed symptom of malfunction. After the candidates are determined, each submodule can then be examined in turn. It is useful to be able to choose the most likely candidate to focus on first so that the faulty parts can be located sooner. We have developed a systematic method for candidate ordering that takes into account the structure of the device and the discrepancy in outputs between the observed and expected values. However, because the same good/bad output pattern of a device always gives rise to the same initial ordering, the method has its limitation. For any device and good/bad output pattern, it is easy to come up with an example on which the method does poorly in the sense that the actual faulty part is in the last place of the initial ordering.

To fix this problem, more dynamic methods, candidate reordering and elimination, were developed. Both methods modify the candidate list as new information becomes available. Reordering moves components connected to the consistent inputs of the current candidate to the front of the candidate list and those connected to the inconsistent inputs to the tail. Elimination removes components which no longer have any non-error-propagating paths to incorrect primary outputs after the current candidate is found not error-propagating. It has been proved that under the single fault assumption, the average number of candidates to be checked is $\Theta(\log n)$, where $n$ is the number of initial candidates. As to the implementation, the VMES system has been successfully ported from our UNIX machine using SNePS-79 and Franz Lisp to TI Explorers using SNePS-2 and Common Lisp. The improvement in preformance is enormous.

An interface for encoding devices in the previously developed method of representation, *i.e.*, using structural templates and instantiation rules, for fault diagnosis was implemented.

This is user-friendly and robust, providing for as few key-strokes from the user as possible. It fills in most of the invariant template, documents the code and stores it in the appropriate file for the user. It was encoded in Franz Lisp on a VAX to begin with, and was transferred to Common Lisp on a TI Explorer.

In an attempt to enable VMES to diagnose sequential circuits, the following were carried out: *(1)* a changed representation scheme, with a proposed representation of transition tables for simple sequential circuits, *(2)* a changed control structure where one not only steps down the structural but also the temporal hierarchy of the device. This changed scheme allows for the usage of shallow reasoning too. Currently, the code for VMES is being re-written for the new representation scheme, and sequential circuit diagnosis is being incorporated in it. Classical techniques in electrical engineering, used to generate tests to find faults in circuits are also being looked into. In order to use these techniques, one has to be able to narrow-down the suspect list sufficiently to contain possible computational explosion. Therefore, work is being done on suspect elimination based on the electrical behavior of the circuit, as opposed to that based on circuit topology.

Deductive reasoning systems, including automatic fault diagnosis, usually make use of vast numbers of rules to infer new knowledge from existing knowledge. Different rules may be at different levels of generality. In particular, a rule may contain a subrule in its consequent. If some instance of the antecedent of the outer rule is satisfied, that instance of the subrule will be asserted in the knowledge base. This subrule has a lower level of generality than its dominating rule. We call this phenomenon the migration of deep to shallow knowledge.

As more specific knowledge emerges from the general knowledge, the speed of inference might become slower if the system tries to use both specific and general rules for the same problem. We need proper schemes for taking advantage of specific knowledge to avoid a search of the similar general rules during the inference, and, in the long run, to realize fast inference. The main idea of shadowing general knowledge by its instances is to exploit the binding information of the free variables of a rule so that this rule can be blocked in future inference if instances of that rule already exist with the same binding information. This shadowing strategy will be more effective on isolating faulty devices in sequential circuits in which previous knowledge is used in later diagnosis.

Natural Language Graphics (NLG) deals with diagram generation driven by natural language utterances. Part of the developed theory has been implemented as a generator program that creates pictures from knowledge structures and as an ATN grammar that creates knowledge structures from limited natural language input. The function of the picture generation program ("**TINA**") as a user interface for a VMES has been demonstrated.

The continuing development of the new version of the Semantic Network Processing System (SNePS-2) has proceeded through the year. During this past year, the mainstream of our system's development has been split in two directions: *(1)* the development and enhancement of the SNePS-2 system itself, and *(2)* making the system operational and maintainable over seven different computer systems.

## 3.2 CANDIDATE ORDERING, REORDERING AND ELIMINATION[1]

### 3.2.1 Introduction

Diagnostic reasoning based on the structural and functional descriptions of a device, usually referred to as the "design model", has been shown to be viable by many AI researchers. This approach reasons from "first principles", *i.e.*, from knowledge of how the device works rather than from knowledge of how it fails. The knowledge needed for such a system is well-structured and readily available when a device is designed, thus avoiding the difficulties of an empirical failure-based diagnosis system in knowledge acquisition, diagnosis capability and system generalization.

Since a model-based fault diagnosis system reasons directly on the structure and function of a device, it usually follows a simple control structure. It starts from the top level of the structural hierarchy of the device and tries to find outputs that produce bad values. After detecting the bad outputs, the system uses structural descriptions to find a subset of components, which might be responsible for the observed symptom of malfunction, at the next lower hierarchical level. This step, known as candidate generation, is a major step in fault diagnosis. This process is then continued with each of the suspicious components in turn until the faulty parts are found. It is desirable to have a diagnostic system which is able to choose the most likely candidate to focus on first so that the faulty parts can be located sooner.

Previous work on diagnosis based on structure and behavior contains suggestions on initial candidate ordering. R. Davis[2] developed a technique called *assumption relaxation* for diagnostic reasoning. The technique uses an ordered list of hypotheses (the kinds of things that can go wrong) and starts to generate candidates under the first hypothesis. If this fails to find the faulty part, it gives up the assumption and considers subsequent hypotheses in order. This approach, in some sense, orders candidates implicitly according to various assumptions, such as single point of failure, bridges and multiple point of failure, etc. However, when more than one candidate is generated using a particular hypothesis, there is no ordering among the generated candidates in Davis's work.

In our previous work, M. Taie *et al.*[3], candidates are explicitly sorted by their *fault possibilities* which are derived from their relationship to bad primary outputs. More precisely,

---

[1]The material discussed in this section is also in the form of a report and several papers. These are:
(1) J. S. Chen and S. N. Srihari. A Method for Ordering Candidate Submodules in Fault Diagnosis. Technical Report TR-8736, Northeast Artificial Intelligence Consortium, 1988.
(2) J. S. Chen and S. N. Srihari. Candidate Ordering and Elimination in Model-based Fault Diagnosis. Submitted to the *International Joint Conference on Artificial Intelligence*, Detroit, Michigan, August 1989.
(3) J. S. Chen, S. N. Srihari and S. J. Upadhyaya. A Heuristic Based Module Ordering for Fault Diagnosis. Submitted to the *19th Annual International Symposium on Fault Tolerant Computing*, Chicago, June 1989.

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

[3]M. R. Taie, J. Geller, S. N. Srihari and S. C. Shapiro. Knowledge based modeling of circuit boards. *Proc. of 1987 Annual Reliability and Maintainability Symposium*, 422–427.

a submodule has a higher fault probability if it contributes to more bad primary outputs. Here we extend this statement to its dual—a submodule has a lower fault probability if it contributes to more good primary outputs.

An important problem which has not been addressed much in model-based diagnosis literature is how a diagnostic system should adjust its focus according to information acquired during the diagnosis. In a typical diagnosis, the system's first conjecture might be incorrect. Nevertheless, the fact that the first candidate is not faulty provides the system with information about the relative probabilities of other candidates. To make use of these kinds of information, we develop candidate reordering and elimination methods. They modify the candidate list during the diagnosis according to newly available information in a way that actual faulty modules are moved forward in the list and modules which are inferred not to be responsible for the incorrect primary outputs are removed from the list.

### 3.2.2 Initial Candidate Ordering

We now describe heuristics for selecting the most likely candidate. They are derived by analyzing the circuit in a manner analogous to that employed by a human diagnostician. Typical examples are used to explain our points. In what follows, an output of a device is said to be bad if for a certain assignment of values to the inputs, the observed output value is different from the expected output value. (An incorrect output is marked with a cross in the figures.) The expected value of an output can be computed by knowing the function of the device. For simplicity, in this section, we concentrate only on the modules A and B in the following figures and assume other modules are well-functioning.

Intuitively, we have the following two heuristics: *(1)* a submodule is more likely to be faulty if it is connected to more bad primary outputs and *(2)* a submodule is less likely to be faulty if it is connected to more good primary outputs. While these rules seem plausible, they need to be further refined. For example, both A and B in Fig. 3.2.1 are connected to the same number of bad primary outputs, *i.e.*, one, and the same number of good primary outputs, *i.e.*, zero. They are equally likely to be faulty according to the above two rules. A closer look reveals that A has two outputs connecting to the bad primary output while B has only one. This makes A more likely to be faulty than B since either of A's two outputs may be responsible for the observed output discrepancy and the probability of A being faulty is the sum of the probabilities of each of its outputs being bad (assuming they are independent). This example shows that we have to consider submodule outputs individually and combine them in some way. Summing up the total number of bad (good) primary outputs connected to each output of a submodule works fine in this case.

The circuit in Fig. 3.2.2 shows that summation is not good enough. Here both A and B have two outputs, and each of them connects to one of the bad primary outputs. The total number of bad primary outputs connected to A's outputs, *i.e.*, two, is the same as that of B's outputs. This time the distinction lies not on the total number but the different number of the bad primary outputs connected to their outputs. The fact that B's outputs cover two bad primary outputs while A's outputs cover only one gives us some evidence

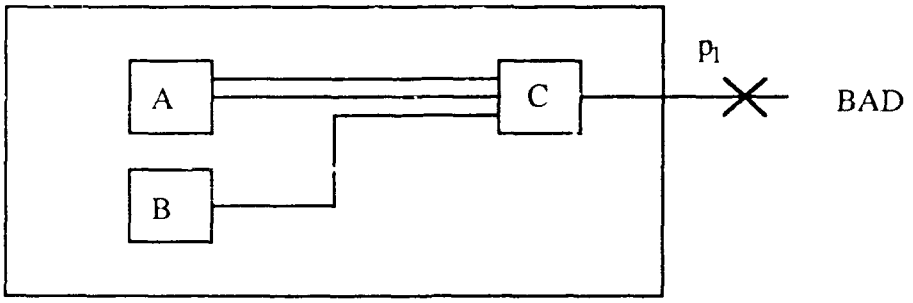


Figure 3.2.1: Submodule's outputs connected to bad primary outputs

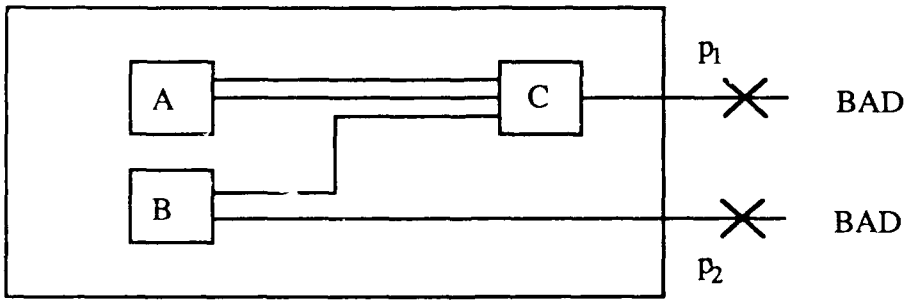


Figure 3.2.2: Bad primary outputs connected to submodules

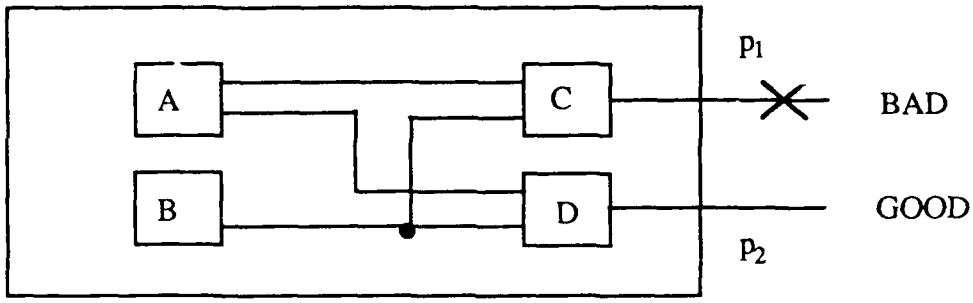


Figure 3.2.3: GI and non-GI outputs

that B is more likely to be the faulty submodule. Thus in addition to summation, we also need to know the union of the sets of bad primary outputs connected to the outputs of a submodule.

Fig. 3.2.3 shows the insufficiency of the previous rules. None of the rules will discriminate A and B since they both cover the same bad and good primary outputs. Nevertheless, the fact that B's only output connects to both the bad and good primary outputs shows evidence of its being good and makes B less likely to be faulty. This makes the necessity to distinguish between the two types of outputs, the first output (the top one) of A and the output of B. More formally, an output of a module is said to be good interactive (or GI) if it connects to any good primary output (*e.g.*, B's output and A's second output); otherwise, it is said to be non-GI (*e.g.*, A's first output). GI outputs are more likely to be good than non-GI outputs.

These examples show that the two heuristics using submodules as references are not adequate. We have to consider the individual outputs of submodules before considering the submodules themselves. In order to combine the results obtained from the individual outputs of a submodule, we need to use additions and set unions with special treatment on GI outputs.

We present in the following an algorithm which assigns a *rank* to a submodule $S$ according to these heuristics. In addition, we also make use of heuristics about the kinds of faults that are more frequently encountered when the device has been functioning for a while. Here we assume that single faults occur more often than other faults.

1. Let $B$ and $G$ be the sets of bad and good primary outputs, respectively. Assume $o_1, o_2, \ldots, o_k$ are the outputs of submodule $S$.

2. For each $o_i$, compute $b_i$ and $g_i$, which are the sets of bad and good primary outputs $o_i$ connects to, respectively. Note that if $g_i = \emptyset$ then $o_i$ is a non-GI output.

3. Compute the *rank* of $S$, $r = (r_1, r_2, r_3, r_4, r_5)$.

- $r_1 = \mid \cup_{i=1}^{k} b_i \mid$, which is the total number of bad primary outputs (without repetition) connected to the outputs of $S$.
- $r_2 = \mid \cup_{g_i=\emptyset} b_i \mid$, which is the total number of bad primary outputs (without repetition) connected to at least one non-GI output of $S$.
- $r_3 = \sum_{i=1}^{k} \mid b_i \mid$, which is the total number of bad primary outputs (with repetition) connected to the outputs of $S$.
- $r_4 = \sum_{g_i=\emptyset} \mid b_i \mid$, which is the same as $r_2$ but with repetition.
- $r_5 = -\sum_{i=1}^{k} \mid g_i \mid$, which is the negative of the total number of good primary outputs (with repetition) connected to the outputs of $S$.

4. Determine whether $S$ is a candidate under the fault assumption we are using according to its rank. For example, $S$ is a candidate under the *single fault assumption* (SFA) if $r_1 = \mid B \mid$, or $S$ is a candidate under the *unidirectional ports assumption* (UPA)[4] if $r_3 > 0$.

After each of the candidate submodules has been identified and assigned a rank, we can now order them according to their ranks. More specifically, we use $r_1$ as the first key, $r_2$ as the second key, etc. The reasons we select these keys are explained as follows. The first key, $r_1$, orders candidates by the number of different bad primary outputs covered by their outputs. The more bad primary outputs a module covers, the more likely its being faulty can explain the observation. The second key, $r_2$, counts the number of bad primary outputs covered by the non-GI outputs of a module since GI outputs have higher probabilities of being good. The third and fifth keys, $r_3$ & $r_5$, are straightforward realization of the heuristic implied by Fig. 3.2.1. The fourth key, $r_4$, serves as a tie breaker for the third key.

To see that the rank assignments actually reflect the heuristics described above, we review the circuit in Fig. 3.2.3 again according to the algorithm. All modules, except D, have outputs connected to the bad primary output, $p_1$, so $r_1(A) = r_1(B) = r_1(C) = r_3(A) = r_3(B) = r_3(C) = 1$ and $r_1(D) = r_3(D) = 0$. Both modules A and C have only one non-GI output connected to the bad primary output, so $r_2(A) = r_4(A) = 1$ and $r_2(C) = r_4(C) = 1$. On the other hand, B and D have no non-GI outputs so $r_2(B) = r_4(B) = 0$ and $r_2(D) = r_4(D) = 0$. Similarly, $r_5(A) = -1$, $r_5(B) = -1$, $r_5(C) = 0$ and $r_5(D) = -1$. Thus we have $r(C) > r(A) > r(B) > r(D)$, which is consistent with the result derived from our heuristics.

[4]Unidirectional ports assumption assumes that the inputs always receive data and outputs always send data, not the other way around. In this paper, we also implicitly assume UPA as we assume SFA.

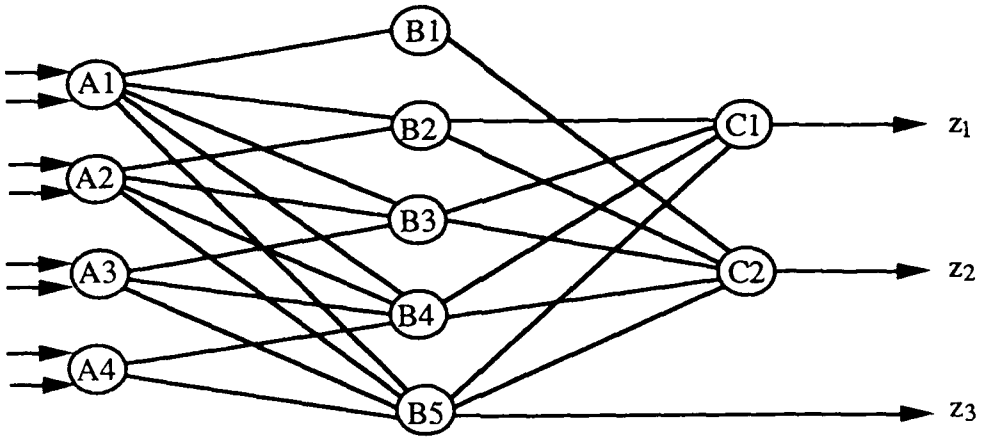


Figure 3.2.4: A single line representation of an example system

### 3.2.3 Analysis of Initial Ordering

In the previous section, we described the initial ordering method according to our heuristics. It is, however, important to obtain some measures to evaluate the effectiveness of our technique of candidate ordering. We first explain a worst case scenario and then report the results of the simulation conducted on an example system.

Let $p$ be the number of primary outputs of a given system and let $m$ be the number of modules in the system. There are $m!$ ways in which the $m$ modules can be scheduled for diagnosis. Since our ordering is based on the connectivity information between the various modules, there are only $2^p - 1$ distinct orderings possible. In other words, faults may occur in a variety of ways, but will reflect at the outputs only in a small number of ways, *i.e.* $(2^p - 1)$ ways. Thus, the probability of selecting the actual faulty module first appears to be small. We now obtain some realistic estimates by means of computer simulations.

We have performed a computer simulation on a system consisting of 11 modules which are organized in 3 levels. The system is described by a single line graph in Fig. 3.2.4, where nodes represent the modules and the edges represent the connections between the various modules. For lack of space, details about the circuit are omitted. Table 1 and 2 list the results of the simulation conducted with single and multiple faults in the modules respectively. The faults (stuck-at-1 and stuck-at-0 selected randomly) are injected into the modules randomly and the set of good and bad primary outputs is determined by applying a single test input (not shown in the tables). The last column of the tables lists the ordering of the modules such that the leftmost module is tested first. We assign a weight of 1 for the leftmost module in the list (highest priority) and a weight of 11 for the rightmost module (least priority). From Table 1, it can be inferred that only three modules will be tested unsuccessfully before the actual faulty module is picked at the fourth attempt, on

| Trial No. | Fault | Module | Bad outputs | Good outputs | Module order |
|---|---|---|---|---|---|
| 1 | s-a-0 | A3 | 2 | 1 3 | A1 C2 B1 B2 B3 B4 B5 A2 **A3** A4 C1 |
| 2 | s-a-1 | C1 | 1 | 2 3 | B5 **C1** B2 B3 B4 A2 A4 A3 A1 B1 C2 |
| 3 | s-a-1 | B5 | 1 | 2 3 | **B5** C1 B2 B3 B4 A2 A4 A3 A1 B1 C2 |
| 4 | s-a-0 | A2 | 3 | 1 2 | A4 B5 A3 **A2** A1 B1 C1 C2 B2 B3 B4 |
| 5 | s-a-1 | B2 | 1 | 2 3 | B5 C1 **B2** B3 B4 A2 A4 A3 A1 B1 C2 |
| 6 | s-a-0 | B1 | 2 | 1 3 | A1 C2 **B1** B2 B3 B4 B5 A2 A3 A4 C1 |
| 7 | s-a-1 | C1 | 1 | 2 3 | B5 **C1** B2 B3 B4 A2 A4 A3 A1 B1 C2 |
| 8 | s-a-0 | A3 | 2 | 1 3 | A1 C2 B1 B2 B3 B4 B5 A2 **A3** A4 C1 |
| 9 | s-a-0 | A1 | 1 | 2 3 | B5 C1 B2 B3 B4 A2 A4 A3 **A1** B1 C2 |
| 10 | s-a-1 | C2 | 2 | 1 3 | A1 **C2** B1 B2 B3 B4 B5 A2 A3 A4 C1 |

Table 3.2.1: Results of simulation with single faults in the modules

| Trial No. | Fault | Module | Bad outputs | Good outputs | Module order |
|---|---|---|---|---|---|
| 1 | s-a-0 | C2 | 2 | 1 3 | A1 **C2** B1 B2 B3 B4 B5 A2 A3 A4 C1 |
| 2 | s-a-0 | B1 | 2 | 1 3 | A1 C2 **B1** B2 B3 B4 B5 A2 A3 A4 C1 |
| 3 | s-a-0 | A1 | 1 2 | 3 | A2 **A1** A3 A4 B5 B3 B4 B2 B1 C1 C2 |
| 4 | s-a-1 | A2 | 2 | 1 3 | A1 C2 B1 B2 B3 B4 B5 **A2** A3 A4 C1 |
| 5 | s-a-0 | B4 | 1 | 2 3 | B5 C1 B2 B3 **B4** A2 A4 A3 A1 B1 C2 |
| 6 | s-a-1 | C2 | 2 | 1 3 | A1 **C2** B1 B2 B3 B4 B5 A2 A3 A4 C1 |
| 7 | s-a-1 | A4 | 1 2 | 3 | A2 A1 A3 **A4** B5 B3 B4 B2 B1 C1 C2 |
| 8 | s-a-1 | B3 | 2 | 1 3 | A1 C2 B1 B2 **B3** B4 B5 A2 A3 A4 C1 |
| 9 | s-a-0 | A3 | 1 2 | 3 | A2 A1 **A3** A4 B5 B3 B4 B2 B1 C1 C2 |
| 10 | s-a-1 | B5 | 2 3 | 1 | **B5** A1 A2 A4 A3 B1 C2 B2 B3 B4 C1 |

Table 3.2.2: Results of simulation with multiple faults in the modules

an average. However, with multiple fault assumption, Table 2 indicates that on an average, only two modules will be tested unsuccessfully before the actual faulty module is picked. Our simulation thus indicates that our heuristic based module ordering method gives a very good initial candidate ordering.

### 3.2.4 Candidate Reordering and Elimination

While the initial candidate ordering looks promising (which can be verified by computer simulation), there is no guarantee that actual faulty components are ordered in the front of the candidate list. In fact, for any device and good/bad output pattern, it is not difficult to come up with a counterexample on which our method does poorly in the sense that the actual faulty component is put at the last place in initial ordering. For example, in Fig. 3.2.3, the culprit might actually be B which is the last candidate (under SFA) according to our initial ordering. To address this problem, we reorder or eliminate candidates whenever some intermediate values are measured.

We now explain the reordering principle by considering the example of Fig. 3.2.3. Assume there is only one fault and B is the actual faulty module. As shown in previous section, the initial candidate list is (C A B). After we check the current candidate C by measuring its inputs and outputs, we will find that the first input of C is consistent with primary inputs but the second input is not since B is the only one at fault. Now we can use this information to shove B to the front of candidate list and A to the tail because B (A) becomes more (less) likely to be faulty. Therefore, the candidate list becomes (B A) and B will be checked next. This shows that B is not the last one to be checked due to candidate reordering although it was at the last place initially. More formally, candidate reordering works as follows. After the inputs of current candidate are measured, candidates connected to its incorrect inputs (with respect to the primary inputs) are shoved to the front of candidate list and candidates connected to correct inputs but not to incorrect inputs are shoved to the tail.

To explain candidate elimination, we use the example in Fig. 3.2.5 and assume E is the only culprit. The initial candidate list is (A B C D E). Since E is the only culprit, A's outputs are consistent with the primary inputs and A is *non-error-propagating* (with respect to the current primary inputs). Module A cannot be responsible for the observed error in $p_2$ nor can B. This is because, B's outputs must go through A to affect $p_2$ but A's outputs have been known to be consistent with current primary inputs. Therefore B can be removed from the candidate list. In contrast to B, Module D cannot be removed at this time due to the existence of a path passing C to $p_2$. This leaves (C D E) as the new candidate list. By the same argument, D will be removed after C's outputs are found to be consistent. As a result, E becomes the only candidate left and the diagnosis at this level can be terminated. In short, whenever the current candidate is found to be non-error-propagating, candidates that no longer have a path to bad primary outputs are removed from the candidate list.

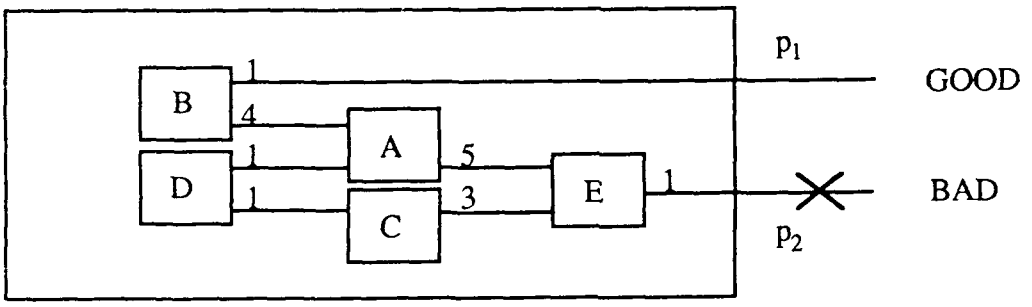


Figure 3.2.5: An example for candidate elimination (numbers are bit widths)

### 3.2.5 Analysis of Reodering and Elimination

To show that candidate reordering and elimination really help shorten the length of a diagnosis, we compute the average number of components that have to be checked before the culprit is found, under single fault assumption (SFA). Note that under SFA, shoving candidates connected to incorrect inputs of current candidate to the front of candidate list is equivalent to removing other candidates as far as the length of diagnosis is concerned. The culprit is guaranteed to be among those being shoved to front since they contribute to some intermediate errors while others don't. Throughout the analysis, the failure distribution is assumed to be uniform. For example, each candidate has equal failure rate and a candidate has a probability of 0.5 of being non-error propagating if it is not faulty.

Let $l(n)$ denote the average length of a diagnosis (number of candidates checked) when there are $n$ candidates. $l(n)$ is given by the following recurrence relation:

$$
\begin{aligned}
l(n) &= \tfrac{1}{n} \cdot 1 && \text{1st candidate faulty: one candidate is checked} \\
&+ \frac{n-1}{n}[1 + \frac{1}{2}\sum_{i=1}^{n-1}\frac{l(i)}{n-1}] && \text{1st candidate intact and non-error-propagating: 0 to } n-2 \text{ candidates may be removed} \\
&+ \frac{n-1}{n}[1 + \frac{1}{2}\sum_{i=1}^{n-1}\frac{l(i)}{n-1}] && \text{otherwise: 1 to } n-1 \text{ candidates may be shoved to front}
\end{aligned}
$$

The above expression simplifies to

$$l(n) = 1 + \frac{1}{n}\sum_{i=1}^{n-1} l(i) \tag{3.2.1}$$

Substituting $n-1$ for $n$ in (3.2.1), we get

$$l(n-1) = 1 + \frac{1}{n-1}\sum_{i=1}^{n-2} l(i) \tag{3.2.2}$$

It follows that

$$\sum_{i=1}^{n-2} l(i) = (n-1)[l(n-1) - 1] \tag{3.2.3}$$

From (3.2.1) and (3.2.3), we have

$$l(n) = \frac{1}{n} + l(n-1) = \Theta(\log n) \tag{3.2.4}$$

If there are $m$ candidates initially ($m$ is bounded by the number of components), the average length of diagnosis using candidate reordering and elimination is $\Theta(\log m)$. This is much better than random or sequential examination whose expected length is $\frac{1}{2}m$.

### 3.2.6 Discussion

The techniques of candidate ordering, reordering and elimination have been implemented in our Versatile Maintenance Expert System (VMES). VMES is a model-based fault diagnosis system which assists inexperienced maintenance technicians in trouble-shooting electronic circuits. As expected, the system shows reasonable improvement in diagnostic performance after these techniques are employed.

The simulation study indicates that the heuristics developed for module ordering work satisfactorily. Although our average case analysis does not depend on the initial ordering of the candidates, presumably a good initial ordering would prevent a diagnosis from going into worst cases. Incidentally, our initial candidate ordering scheme can be easily augmented to include other information useful for a system to produce a better initial ordering. All we need to do is introduce new keys representing the new information in appropriate positions. For example, if we have information about the failure rate and test cost of each component type, we can simply add them as, say, the second and fourth keys.

We view diagnosis as a progressive process. Until the culprits of a malfunctioning device are found, a diagnostic system must be able to produce a new answer and confirm the answer. The system is allowed to make mistakes but should not only recover from them but also discover useful information from them. The candidate reordering and elimination techniques are motivated by these ideas. They are somewhat limited due to the need for measuring values at intermediate points of the diagnosed device. Our future work will include the development of similar "self-adjusting" search methods that are based on other

kinds of information which can be acquired during diagnosis. The behavior of the diagnosed device after some boards are replaced or swapped, and its responses to different primary (external) inputs, are currently under consideration.

## 3.3 SELECTION OF A REAL DEVICE

This year saw an advance towards VMES being a real-time expert system: the selection of a real test device. The advantage of working with a real test device is that the applicability of our ideas could be tested out for a practical situation.

### 3.3.1 Printer Buffer Board

The Heathkit printer buffer board model SK-203 (see Fig. 3.3.1) was selected as the device on which to test the concepts developed in the VMES project. The board, assembled locally, is a microprocessor controlled buffer that stores data received from the computer before sending it on to the printer. It was selected for the range of sub-devices it incorporates and the complexity of its operation.

The board has a controlling 8 bit CMOS microprocessor, an input serial and an input parallel port, an output serial and an output parallel port. On board are 512 kilobytes of RAM and 2 kilobytes of ROM containing programs that control operation of the device. The board has combinational components such as address decoders, port controllers and read-write controllers. The microprocessor, memory, read and display control latches are some of the sequential circuits to be found. The board also has analog circuit modules such as power supply and filter.

The structural hierarchy of the board is fashioned after the functional units in it (see Fig. 3.3.2). At the top level, it is a printer buffer board. At the next lower level, it contains a display system, a keyboard input, a control unit, memory and the ports. Each of these can be further broken into lower levels. For example, the display system in turn consists of LED display units, latches to hold the display values and interconnecting wires. Though a further level of hierarchy could be brought out in the analog devices, (*i.e.*, at the resistor-transistor level), we don't concern ourselves with analog devices at this point. The device can be put into the following temporal levels : at the gross level of buffer behavior, at the machine instruction level of the controlling CPU, at the clock cycle level and at the gate delay level.

The printer buffer board is structurally diverse and offers ample opportunities for model based deep reasoning. Some of the failures of the device can be captured in shallow rules which can be used for fast diagnosis. Some of the shallow rules, supplied by the device manufacturer are:

If the unit is inoperative, check power supply connections;

If the display is random, ICSA has been wrongly installed;

If display fails to indicate free memory, certain jumpers are wrong.

CPU failure, defective memory and port controller failure are some of the other possible functional failures which should be taken care of by deep reasoning.

The printer buffer board is being implemented in the current representation scheme. The representation scheme is being augmented to accommodate sequential circuits. The

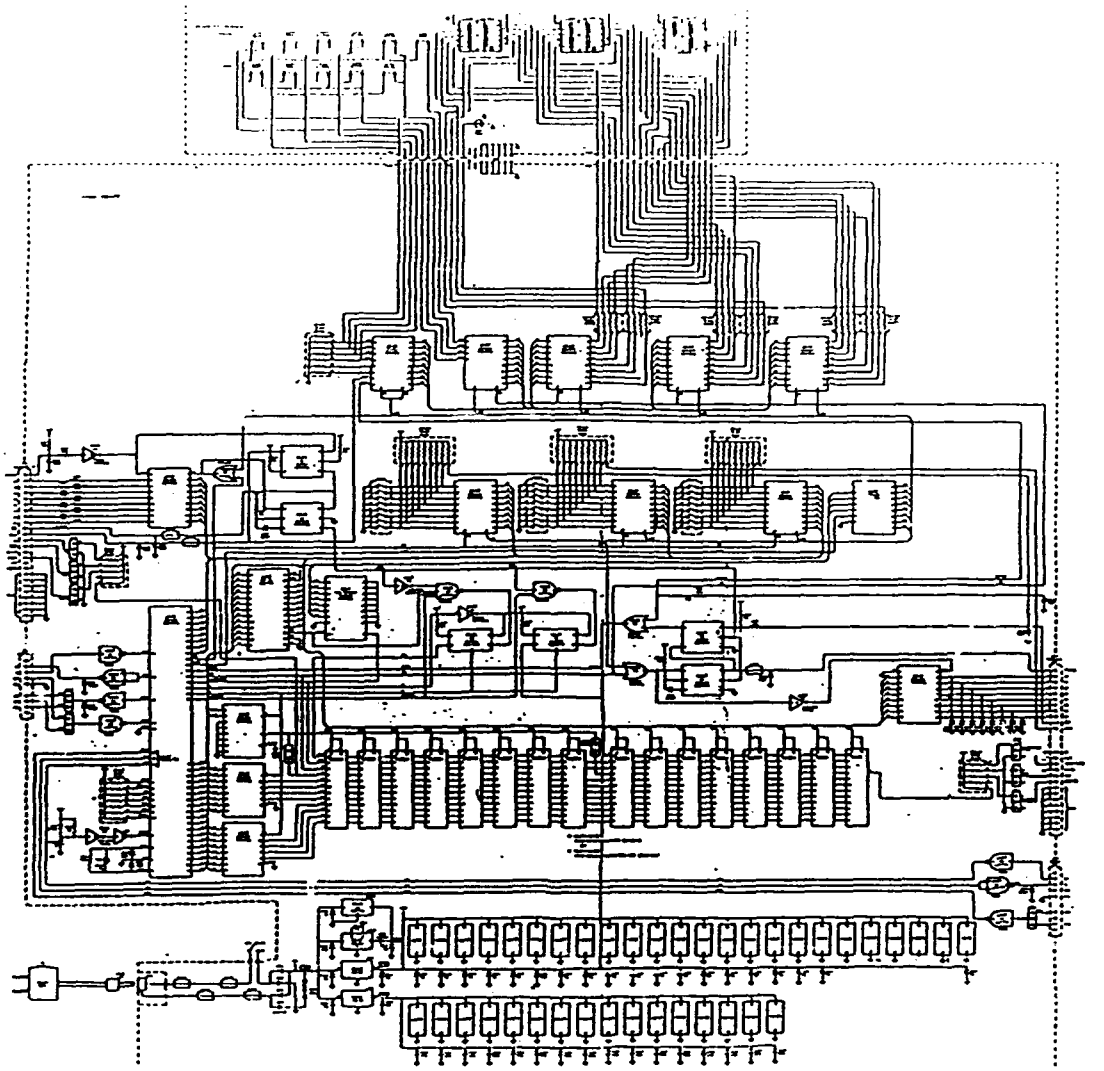

Figure 3.3.1: The Printer Buffer Board

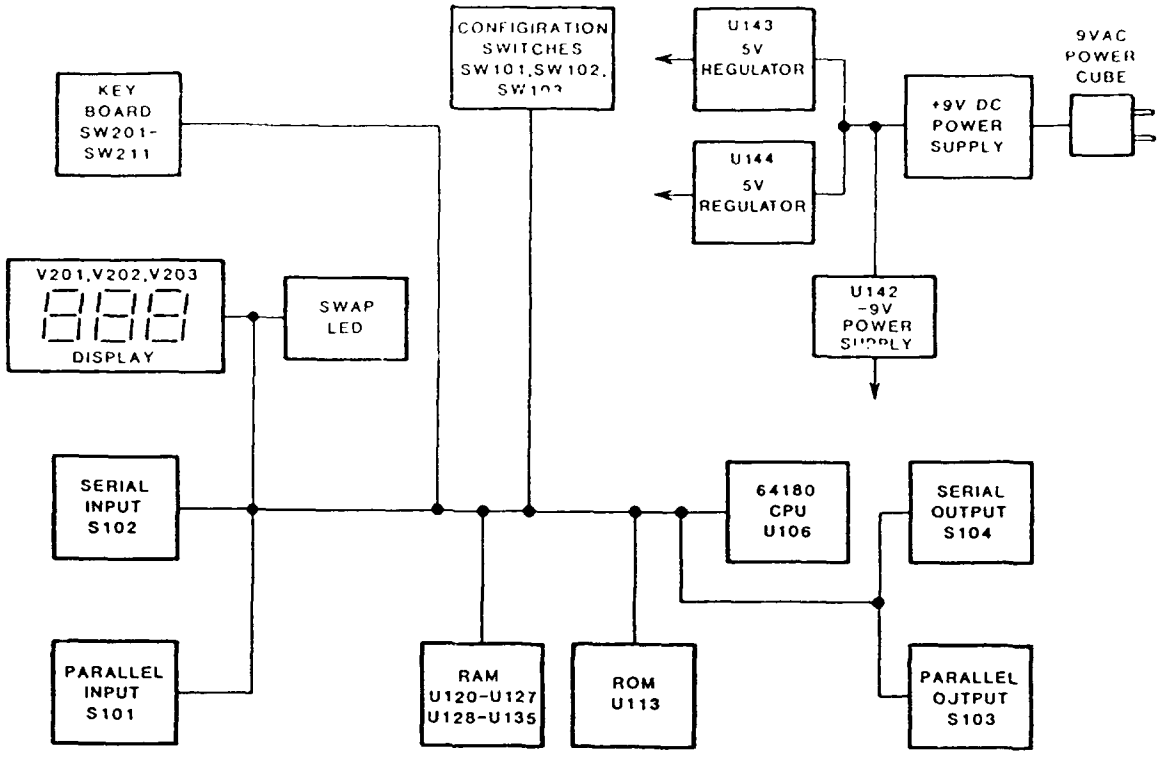


Figure 3.3.2: Logical Units of the Printer Buffer Board

next goal is to extend VMES to diagnose sequential circuits and test it on the printer buffer board.

## 3.4 REPRESENTATION AND DIAGNOSIS OF SEQUENTIAL CIRCUITS

### 3.4.1 Introduction

In literature, constraint based representation of structure and behavior has been used to localize faults in combinational circuits.[1] 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.[2] Expectation violation was used for candidate generation in combinational circuits. It was proved that diagnostic procedure applied to combinational circuits was indiscriminate when applied to sequential circuits. Single stepping was suggested as a divide and conquer strategy to localize faults in sequential circuits. Structural detail was however used to make candidate generator discriminating, at great cost.

We propose that state transition relation should be represented and utilized for diagnosis of sequential circuits and not the input-output relation as in the case of combinational circuits. We outline an algorithm to represent complex sequential circuits so as to facilitate diagnosis and suggest a control structure to single step down the structural and temporal hierarchies of the circuit.

### 3.4.2 Representation of Circuits with State

We suggest the following representation algorithm for circuits, along temporal hierarchy. The idea is to start from the most basic blocks and build step by step, 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 the truth-table or boolean expression. For most purposes the second representation will be adequate.

*(2)* Flip-flops are represented at two levels: the clock period level and at the transition diagram or state table level.

*(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. Again, if the board is functionally sequential it is represented using state diagrams. If it is combinational in behavior it is represented using input-output relations.

---

[1] R. Davis, Diagnosis Via Causal Reasoning: Paths of Interaction and the Locality Principle, *Proc. of AAAI-83*, pp 88–94, AAAI, August, 1983.

[2] Walter Hamscher, R. Davis, Diagnosing Circuits with State: An Inherently Underconstrained Problem, *Proc. of AAAI-84*, pp 142–147, AAAI, August, 1984.

The different levels of time representation chosen are however expressible as integral multiples of the most basic of the levels. Also note that the two levels suggested may be the same for some device, in which case, only one level of representation is used. This scheme is applicable in most general cases of synchronous circuits.

### 3.4.3 State Transition Representation for Sequential Components

Unlike in combinational circuits, output of sequential circuits depends on not only the input but also the state of the device. The state of the circuit is represented by its previous output. Hence, representation of sequential components should provide for the storage of the output of the circuit. This storage variable is initialized at the start of the test, to the reset/initialization value. For every successive output, the value of this variable is taken as the state, the current output is computed and checked for correctness vis-a-vis the input and state. The variable is now updated with the current value. Thus, output variable is used in diagnosis.

### 3.4.4 Control Strategy

The current control strategy in VMES is:

REPEAT
Step down the structural hierarchy for each candidate;
Generate candidates by constraint elimination;
Order candidates by the likelihood of being faulty, according to some heuristic (optional);
UNTIL fault found or further stepping down is impossible.

The above algorithm cannot be used per se for sequential circuits. Generating candidates by constraint elimination fails to be very useful in sequential circuits because of the existence of 'memory' component. Single stepping through temporal hierarchy is found to help the situation. In the new scheme of diagnosing sequential circuits, the following are the steps:

REPEAT
Single step once through temporal hierarchy (if possible);
Single step once through structural hierarchy ;
Generate candidates by constraint elimination:
Order candidates according to some heuristic;
Shrink the candidate list by asking for more information;
UNTIL fault diagnosed or basic level of structural and temporal hierarchy reached.

Note that the representation scheme and control strategy are designed to be compatible. Also note that stepping down the temporal hierarchy may not always be possible, because, a device may have only one level of temporal representation.

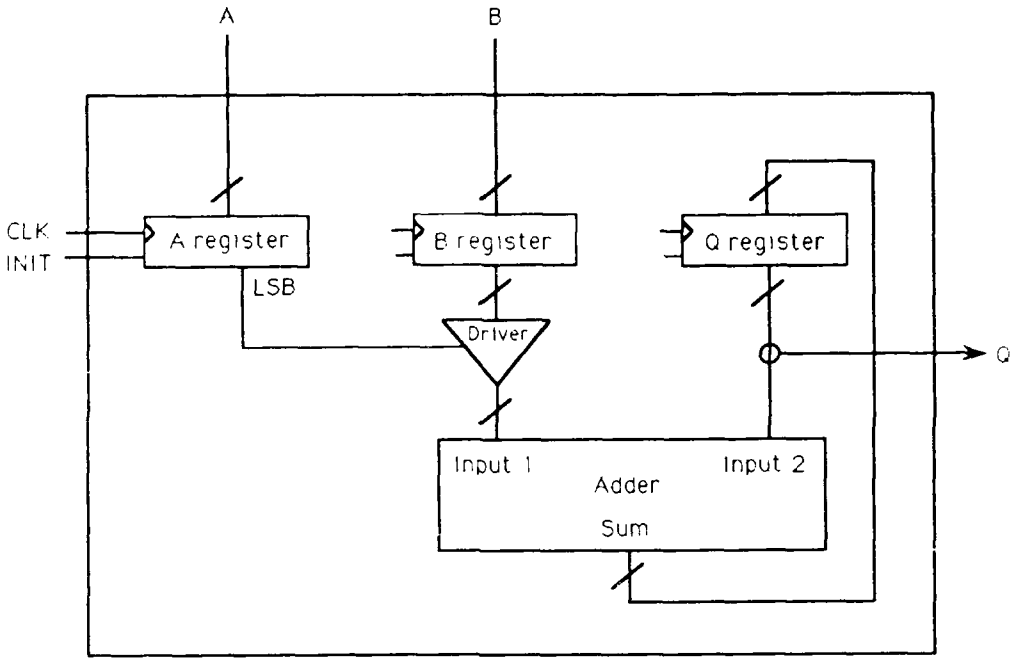


Figure 3.4.1: 4-bit Sequential Multiplier

### 3.4.5 *Candidate Generation as Applied to Circuits with State*

Consider the 4-bit sequential multiplier[2], shown in Fig. 3.4.1 It has two 4-bit registers A and B to contain the multiplier and the multiplicand and an 8-bit accumulator Q. Registers A and B are loaded when the INIT signal is provided. On each clock pulse, register A is shifted right and register B is shifted left. Register Q contains the product after four clock pulses.

Suppose the inputs loaded into the multiplier are 6 and 9; and the output observed is 58 instead of 54. The classical approach is to convert the sequential circuit into equivalent combinational circuit by replicating the circuit behavior over five clock pulses (or time slices). Tracing back from the violated expectation output, all five components are listed as potential candidates since all of them contribute to the output. To find out if the candidates are consistent, each of their behaviors is suspended in turn to check if its removal is consistent with the violated output. Single fault assumption is made, so that the constraints imposed by the candidate can be removed in all time slices. This approach eliminates only register A from being a consistent candidate, since no sequence of 1's and 0's can result in a product of 58.

At this point, single stepping of the circuit is suggested. If we could observe that the output of the Q register is 0,1,20,57 and 58 instead of 0,0,18,54 and 54, we have *(1)* a strictly combinational circuit in each clock pulse; *(2)* four sets of tests to help us diagnose.

This step helps rule out register B, because, the Q register output 1 cannot be explained by suspending constraints imposed by it. As to the other components, one can only conjecture that their pattern of misbehavior is as shown in the following table[2].

| Driver | | | |
|---|---|---|---|
| IN | CONTROL INPUT | ACTUAL OUTPUT | EXPECTED OUTPUT |
| 9 | 0 | 1 | 0 |
| 18 | 1 | 19 | 18 |
| 36 | 1 | 37 | 36 |
| 72 | 0 | 1 | 0 |

| Adder | | | |
|---|---|---|---|
| INPUT1 | INPUT2 | ACTUAL OUTPUT | EXPECTED OUTPUT |
| 0 | 0 | 1 | 0 |
| 18 | 1 | 20 | 19 |
| 36 | 20 | 57 | 56 |
| 0 | 57 | 58 | 57 |

| Q Register | | |
|---|---|---|
| INPUT at t | ACTUAL OUTPUT at t+1 | EXPECTED OUTPUT |
| 0 | 1 | 0 |
| 19 | 20 | 19 |
| 56 | 57 | 56 |
| 57 | 58 | 57 |

The diagnosis in our scheme is as follows: *(1)* The 4 bit multiplier is found to be faulty. Given inputs 6 and 9, its output is 58 instead of 54. *(2)* Stepping down the temporal hierarchy, we observe that the output of the multiplier are 0,1,20,57 and 58 in the five time slices. *(3)* Stepping down the structural hierarchy, we have 2 4-bit registers, an 8-bit register, a driver and an adder instead of a 4-bit multiplier. *(4)* By constraint elimination, we generate the candidates driver, adder and Q register, as before. *(5)* We order the suspects. *(6)* We ask more information of the user, to eliminate suspects. For example, suppose during time t1 we observe that with the control input 0 the output of the driver is 1. The driver can be immediately concluded to be a faulty device. *(7)* The loop is repeated, if necessary. Suppose Q register was found faulty and the Q register was actually an assembly of 8 discrete flip-flops. From the violated expectation we know that the Q register summed up to 19 instead of 18. Stepping down the structural hierarchy for the same temporal granularity, we find that the flip-flop corresponding to the LSB has developed an intermittent fault.

Note that the behavior of registers and adder was represented at just one level because, the time granularity of the overall behavior was at the clock pulse level itself. Hence, single stepping down temporal hierarchy at the level of adder and registers was not possible.

### 3.4.6 Discussion

From the above explanation, the following can be concluded:

*(1)* the advantage of checking sequential circuits for state-transition violation instead of output-violation is clear. With observed violation of expectation during any time slice, we can unequivocally pin down the corresponding device as being faulty, irrespective of the behavior of other devices. For instance, in the above example, if only the output of the Q register was observed at time t3, and it was found to be 19, it could possibly be that the earlier output was 1 and hence Q register was functioning properly. Or, suppose the output of Q register was measured to be 0 at the beginning of time slice t2, and its input was measured to be 18. However, at t3, its output was found to be 19. Clearly, the Q register has shown a transition-violation and is hence faulty.

*(2)* non-intermittency assumption need not be made, *i.e.*, we need not have to assume that faults in the devices are non-intermittent. We need not have to insist that the behavior deduced for a candidate be consistent across all time slices. The tables derived above would be unnecessary. In the example above, the Q register behaved normally in time slice t1. However, during t2, an intermittent fault developed in the register, resulting in violation of state-transition-expectation. This has however been successfully diagnosed in our system.

*(3)* single-fault assumption need not be made. By stepping down structural hierarchy for the same time granularity, we were able to isolate devices as being faulty, without assuming that a failing device could do so in only one way. We were not required to look for any consistency of behavior of the devices, across time slices.

Note that *(2)* and *(3)* above are more a consequence of complete state visibility. However, the representation scheme and diagnostic strategy we propose provide towards deriving those advantages.

### 3.4.7 Conclusion

We have proposed an algorithm to represent complex sequential circuits so as to facilitate diagnosis and suggested a control structure for single stepping down the circuit. The representation and control strategies help exploit the advantages of complete state visibility. We have also demonstrated the suitability of using state transitions instead of input-output relations, as constraints for sequential components in circuits.

### 3.4.8 Related Ongoing Work

We are looking into classical techniques in electrical engineering, used to generate tests to find faults in circuits. In order to use these techniques, one has to be able to narrow-down the suspect list sufficiently to contain possible computational explosion. Therefore, work is being done on suspect elimination based on the electrical behavior of the circuit, as opposed to that based on circuit topology.

## 3.5 SHADOWING DEEP KNOWLEDGE BY INSTANCES

### 3.5.1 Introduction

This section describes a new idea for the VMES project which concerns the speed of inference during diagnosis.

AI systems are generally divided into two categories: deep systems and surface systems. Surface systems are those that have no underlying representation for such fundamental concepts as causality, intent, or basic physical principles. They use shallow knowledge that simply associates input states with actions in a direct manner. By contrast, deep systems try to represent deep knowledge which enables them to make deductions from a compact collection of fundamental principles. A surface system is preferred for a task of modest difficulty, and it is faster than deep systems. However, a deep system is needed to perform complex tasks for which sophisticated causal chains are indispensable, despite high startup costs.

This consideration naturally leads to combining deep and surface systems into a multi-level system for the purpose of taking benefits from both of them. Among several issues in this multi-level system, we consider the problems of how multiple levels of representation can be exploited, and how a surface model can be compiled from a deep model to generate a faster system.

These issues have been widely discussed in the domain of an automatic fault diagnosis system which locates faulty parts in a malfunctioning device. Traditional diagnosis systems which use shallow model are built on empirical rules that associate observed symptoms with possible fault hypotheses. Although considered successful, these systems have significant drawbacks, one of which is the difficulty of applying them to different devices because there is almost no capability of system generalization. To overcome this and give versatility to the system, recent systems are focusing on the ability to adapt to different devices without extensive knowledge engineering, and on the capability of diagnosing a wide range of common faults. Generic domain knowledge represented by structural and functional descriptions are suggested as a solution to the problems caused by empirical rule based diagnosis systems. But still not much work has been done when different levels of knowledge are present in the knowledge base. What we are trying to do is, whenever several pieces of knowledge can be applied at the same time, we apply the most specific one.

The motivation for dealing with multi-level knowledge basically comes from observing the behaviors of deductive reasoning systems. A deductive reasoning system usually has two kinds of assertional knowledge in the knowledge base (KB): rules and facts. The rules are represented in implicational form. Typically, they express general knowledge about a particular area. The facts represent specific knowledge relevant to a particular case. Although rules are considered to be general knowledge (or deep knowledge), levels of generalities differ from one another. (For example, the rule $A(a) \Rightarrow B(a)$ is more specific than the rule $\forall x\{A(x) \Rightarrow B(x)\}$.) In particular, a rule which contains another subrule

inside can lead to the assertion of the subrule if its antecedent is satisfied with known instances. This subrule has a shallower level of generality than its dominating rule, so the former is more specific than the latter. We call this phenomenon the automatic migration of deep to shallow knowledge.

As more shallow knowledge is migrated from deep knowledge, the speed of inference might become slower if the system investigates both levels during inference. We only want to look at shallow knowledge if possible. So we need a proper way to take advantage of shallow knowledge to avoid duplicated search of similar rules, and, eventually, to realize fast inference. The main idea of shadowing deep knowledge is that after instances are found for the general rule, we check if those instances are acceptable by filtering them out with binding information for free variables in the general rule. Should any of those instances be accepted, the general rule will be shadowed.

### 3.5.2 Automatic Migration of Deep to Shallow Knowledge

The major task of the reasoning system is to derive new implied facts from existing rules and facts in the KB which are known to be true. The deduced information will be asserted into the KB. Sometimes, however, what the system derives is more than what was requested. This extra information can be useful to enhance system performance.

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

R1 : $\forall R\{transitive(R) \Rightarrow \forall x, y, z\{R(x,y)\&R(y,z) \Rightarrow R(x,z)\}\}$

R1 reads: if $R$ is a transitive relation, and *R(x,y)* and *R(y,z)* hold, then *R(x,z)* holds. R1 is an embedded rule in which the consequent happens to be another rule. In order to show how the reasoning system automatically deduces new facts, let us suppose the KB contains the following facts as well as R1.

F1 : *transitive(supports).*
F2 : *supports(a,b).*
F3 : *supports(b,c).*
F4 : *supports(c,d).*

We want to infer *supports(a,c)* from this KB. Basically the inference proceeds with the help of a pattern matcher which returns all matched instances in KB with requested one. In backward chaining inference like this example, the request is matched with either facts or the consequent part of rules. After *supports(a,c)* and *R(x,z)* are matched and $R$ is substituted by *supports*, the pattern matcher goes further to satisfy the antecedent of R1. Since F1 is asserted, the system infers a new rule R2 and asserts it into KB.

R2 : $\forall x, y, z\{supports(x,y)\&supports(y,z) \Rightarrow supports(x,z)\}$

The inference continues by looking at F2 and F3, and accepts *supports(a,c)* as deduced. Finally *supports(a,c)* is asserted into the KB. What we tried to infer was *supports(a,c)*, but

in addition to that we also get an instantiated rule R2 by side effect. Although both R1 and R2 belong to the general knowledge category, they have different levels of generalities. Since R2 is clearly a derived instance of R1 with a substitution {*supports*/R}, R2 can be regarded as more specific than R1. This is one example of automatic migration of deep to shallow knowledge during inference. Although R2 is not initially required to be deduced, we can get some benefits from R2, especially on the efficiency of the system.

The issue now is how to shadow deep knowledge by its instances (shallow knowledge) during each stage of the inference without losing anything and without considering deep knowledge. In other words, we want to take advantages of shallow knowledge over deep knowledge to speed up the entire inference process because it takes less time to deal with a specific rule than with a general one. After R2 is asserted in the previous example, R1 must be blocked (or ignored) in the next inference which also unifies *V1* of R1 to *supports*. Actually we cannot infer anything from R1 if we cannot infer it from R2 as long as *supports* is involved.

### 3.5.3 SNIP: SNePS Inference Package

SNePS is a knowledge representation/reasoning system using a semantic network formalism. Since rules and facts are represented by semantic networks, the reasoning of SNePS is also performed by exploiting the nature of the semantic network, which contains nodes and arcs. The reasoning part of SNePS is called SNIP. The current version of SNIP has several interesting features.

- 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 an old version of SNIP)
- There ia a smaller set of processes and the types of processes are limited to the types of nodes found in the network. SNIP has 3 types of processes, that is general proposition node process, rule node process, and 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, and **requests** contain 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. The *known-instances* register collects instances of this node which are known to be true (both positive and negative). The *reports* register is the collection of reports received from other node processes. Finally *requests* register is the collection of requests received from other node processes

To show how SNIP works, we first translate R1 and F1 through F4 into SNePSUL (SNePS User Language) form which are shown below.

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

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

SNePS builds network structures for these rules and facts. Figure 3.5.1 shows semantic network structures for this example. A node name followed by ! symbol indicates that this node is asserted at the top level.

In order to infer *supports(a,c)*, **deduce** command is used as (**deduce agent a verb supports object c**). SNIP assigns a process to each SNePS node when it is involved during the inference initiated by deduce. 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 an antecedent or consequent part of the other node. Those channels made by rules are used for implementing rule chaining. During SNIPS backward inference of *supports(a,c)*, an instance of P5 is asserted into KB as.

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

Now suppose we want to infer another implied fact, *supports(b,d)* by (**deduce agent b verb supports object d**). SNePS builds **M8** for this node. The inference procedure now becomes more complicated due to the inclusion of R2 as well as R1.

Initially a user process is created and invokes process **M8**. **M8** is matched with both **P4** and **P8**. A request is sent from process **M8** to process **P4** with a substitution {**b/V2, d/V4, supports/V1**}. At the same time process **M8** sends another request to process **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**, process **F4** sends the request to process **P5** with {**b/V2, d/V4, supports/V1**}. **P8** is also the consequent part of **M7!**, so process **P8** sends the request to process **M7!** with {**b/V2, d/V4**}. These steps of sending requests proceed in parallel. **P5** sends the request to its antecedent processes **P2** and **P3**. Process **P2** and **P3** are activated in the same manner as in the previous inference. Now consider **M7!** after getting the request from **P8**. Since **M7!** is an asserted rule node, requests are sent to its antecedents **P6** and **P7** with {**b/V2, d/V4**}. Since **P6** and **P7** are matched

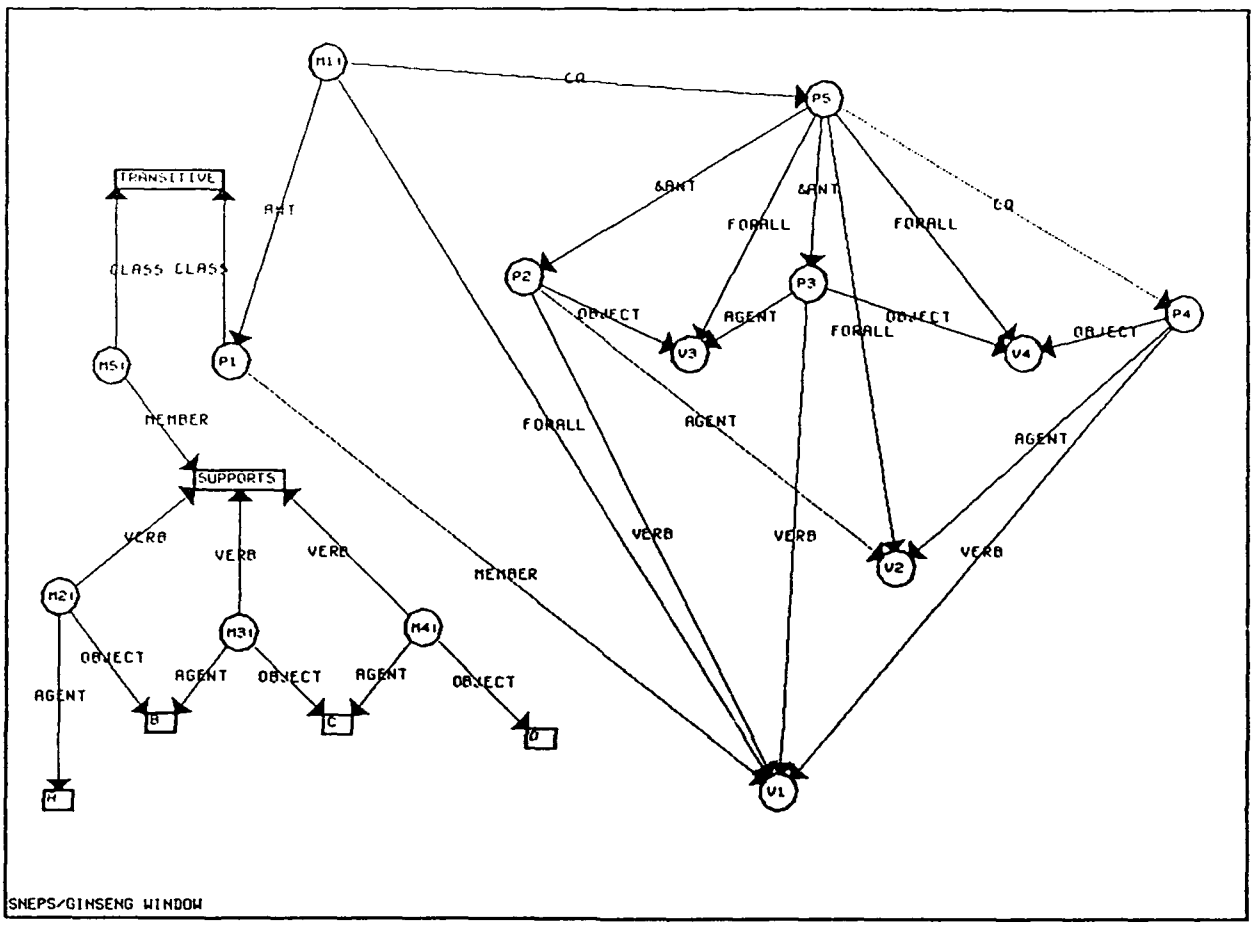


Figure 3.5.1: SNePS nodes for R1, F1 through F4

with M3! and M4!, respectively, process M7! gets reports from process P6 and P7, and then send them back to P8. Process P8 sends the reports back to process M8 which then sends them to the user process. Finally M8 is asserted as M8!.

Notice that while process M7! is active with its antecedent processes, process P5 is also doing the same thing that was done in the previous inference of *supports(a,c)*. We can regard P5 as a redundant process because we cannot expect anything more in P5 than in M7! as long as binding information includes {supports/V1}. Actually this redundant process activation resulted from the inability to recognize M7! as an instances of P5.

### 3.5.4 Shadowing Deep Knowledge by Its Instances

We need to modify SNIP so that the redundant process activation is prevented systematically and safely. This means we improve the efficiency of the system without losing any necessary information. A scheme of shadowing is designed by considering the binding information for free variables in the general rule. As mentioned in the previous example, after *supports(a,c)* is deduced, R2 (M7! in SNePS) is asserted as a side effect. When M7! is being asserted, SNIP sets the *known-instances* register of process P5 to M7!. The exact contents of *known-instances* register will be:

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

This says M7! is a positive instance of P5 with the substitution of {supports/V1}. Notice that *known-instances* not only has the name of the instance, but also includes the binding information for free variables, which is necessary to verify the appropriate instances.

The next inference is *supports(b,d)*. Refer again the process activation in the previous section. When process P4 is sending a request to P5 with the substitution of {supports/V1, b/V2, d/V4}, the *requests* register of process P5 is set to:

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

When P5 is activated with the above request information, we first have to check if P5 has known instances by looking at the *known-instances* register. Since M7! is already there, the next step is to verify that M7! is a useful instance in this inference. In this filtering process, the binding information for free variables in both registers plays the key role. Eventually M7! is accepted as a useful instance because the substitution for free variable of P5 in *requests* is identical to that in *known-instances*. Therefore we stop the activation of process P5. Process M7! will take care of everything.

The situation changes, however, if the query becomes (deduce agent b verb *r object d) in which *r denotes a SNePS variable. (We call this type of question a wh-question which means the question of when, where, what, etc.). The contents of *known-instances* remains the same as before, but the substitution in the *requests* register no longer has a binding for V1.

Actually the nature of the query is to find out all possible relations between *b* and *d*, not necessarily '*supports*'. So although M7! is found as an instance of P5, we should continue searching for other instances of P5 in order to get all possible answers.

An algorithm for shadowing deep knowledge is set by recognizing the different situations illustrated above. The main idea is that if a pattern rule node, which has free variables, has instances, we compare the binding information for free variables in the *known-instances* register with that in the *requests* register in the pattern node process. If two bindings are equal, the process which is assigned to the pattern rule node is blocked, and if there is no bound value for free variables in the *requests* register (in case of wh-question), the system just continues the inference as before with both rules.

### 3.5.5 Further Work

A demonstration for the transitive relation example shows that the inference speed is nearly doubled by the shadowing scheme. This shadowing method can be applied to any deep system, including versatile expert system, which uses deep knowledge to enhance the system performance. It will be more effective in diagnosing sequential circuits because there should be cases when rules for sequential circuits will be re-applied with the previous output and current inputs.

## 3.6 INTERFACES

### 3.6.1 The Device Interface: Frontend

Frontend is an interface to encode devices for diagnosis in VMES. It is user-friendly and is designed to encode circuit devices in Mingruey Taie's representation,[1] *i.e.*, using structural templates and inheritance rules. Requiring as few keystrokes as possible, this interface has the following features:

The interface asks the user questions to elicit the details of the device being represented. The questions go through the various segments of device representation in order. Thus, no section of the necessary code is missed. The segments of representation code that do not change from device to device are automatically filled in.

The interface is robust, *i.e.*, most of the questions asked by it not only specify the nature of the answer expected, but also reject unacceptable answers. For example, if a fixed point number is typed in where an integer was expected, the question is repeated till some integer is entered.

Files are named, opened and closed automatically. Mingruey Taie's convention is followed for naming the files. When a device is completely coded, the name of the file where the representation of the device can be found, is displayed for convenience.

Documentation of the files is automatic. Documentation includes the last name of the author, the date of encoding and a brief explanation as to the nature of the contents of the file. This information is patched to the beginning of the file created. The code generated is pretty-printed. This makes reading the code easier.

FRONTEND was written in FranzLisp on a VAX and was later transferred to Common Lisp on a TI Explorer. The code is modular and accommodates future extensions to representation easily.

### 3.6.2 Natural Language Graphics[2]

Natural Language Graphics (NLG) deals with diagram generation driven by natural language utterances. This investigation applies the methods of declarative knowledge representation to NLG systems. Declarative knowledge that can be used for display purposes as well as reasoning purposes is termed "Graphical Deep Knowledge" and described by supplying syntax and semantics of its constructs. A task domain analysis of Graphical Deep Knowledge is presented covering forms, position, attributes, part, parts, classes, reference frames, inheritability, etc.

---

[1] Mingruey R. Taie, Representation of Device Knowledge for Versatile Fault Diagnosis, Technical Report 87-07, Department of Computer Science, SUNY at Buffalo, May 1987.

[2] This section is a condensation of a paper by J. Geller, "Natural Language Graphics for Human Computer Interaction". Submitted to the *International Journal of Man-Machine Studies.*

Part hierarchies are demonstrated, and criticism of traditional inheritance-based knowledge representation formalisms is derived from this finding. The "Linearity Principle of Knowledge Representation" is introduced and used to constrain some of the presented knowledge structures. The analysis leading to Graphical Deep Knowledge also results in the description of two fundamental conjectures about knowledge representation.

The Gricean maxims of cooperative communication are used as another source of constraints for NLG systems. A new maxim for technical languages is introduced, the "Maxim of Unnecessary Variation". It is argued that common symbolic representations like circuit board diagrams have not yet been described in literature by explicit feature analysis, and that this in necessary to give a system knowledge about the meaning of the diagram it is displaying.

Part of the developed theory has been implemented as a generator program that creates pictures from knowledge structures and as an ATN grammar that creates knowledge structures from limited natural language input. The function of the picture generation program (TINA) as a user interface for a VMES has been demonstrated. An older version of TINA incorporates a module of "Intelligent Machine Drafting" (IMD), a completely new subfield of AI that has been introduced in this research and that relates to Computer Aided Design (CAD). The IMD program does layout and routing for the members of a simple class of functional circuit diagrams based on a policy of symmetry and equal distribution over the available space. This layout/routing is based on cognitive requirements as opposed to physical requirements used by CAD systems.

## Representational Constructs of Graphical Deep Knowledge

*Form Knowledge*

A number of different scientific subfields and fields have been interested in the representation of forms. Among these are computer vision, computer graphics, and imagery, but also solid modeling computer aided design (CAD) and character recognition. We argue[3] that no representation in any of these fields satisfies the requirements for graphical deep knowledge. These requirements are:

- The representation should be projectively adequate.
- The representation should be deductively adequate.
- The representation should be based on conceptual primitives that seem natural to the human observer.
- The representation should support relations between primitives which are natural to humans.
- The representation may contain redundant information.

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

To fulfill these requirements a representation that consists of basic forms (icons) and asserted relations is used. The basic forms are (supposed to be) meaningful to human observers. Every basic form is represented as a procedure that has three properties. *(1)* The procedure consists of calls to graphics primitives. *(2)* Executing a procedure of the name <name> results in the drawing of an object that is described by <name>. *(3)* The procedure <name> is accessible as a concept in the knowledge representation system, *i.e.*, it functions simultaneously as a node in a semantic network. The representation of a basic form is therefore projectively adequate and also a conceptual unit. Relations between icons are represented propositionally.

The SNePS system is used in the following way to accommodate the described form representation. The name of every basic form in the system is a base node in the SNePS semantic network. The SNePS inference machine treats it as a conceptual unit and permits reasoning about it. At the same time every SNePS node is also an uninterned LISP atom. This atom refers[4] to a LISP function made up of calls to graphics primitives from a LISP graphics package. Objects and forms are separate nodes, linked by an asserted proposition. This conceptual separation of forms and objects makes it possible to associate a form with a class of objects, instead of a single object.

### *Part Hierarchies*

Part hierarchies have been of fundamental importance in a number of different areas of artificial intelligence. Knowledge representation has dealt with them as well as hardware modeling in maintenance and research in computer vision.

Our interest in part hierarchies is motivated by the need for a method to decide *"what content"* to put on the screen of an NLG system and *"how to organize it,"* to be optimally useful to a viewer. In KBUIMS (knowledge based user interface management system) design this complex of problems has been referred to as "presentation planning".

Part hierarchies permit a strategy to decide what to show and how to avoid information overload: don't show all the parts of a requested object. If a simple display is expected, just show an integral object. If a more informative display is expected, then show the integral object with its parts. More generally, control the complexity of a display by selecting the number of levels of the part hierarchy that are shown on the screen.

### *The Class Hierarchy*

In our theory a non-tangled class hierarchy is used for standard downward inheritance. However, we also supply a limited upward inheritance facility. We find justification for this in the psychological research on categorization. The cognitive science literature reports three different approaches to categorization the *classical* approach, the *prototype* approach and the *exemplar* approach. The classical approach has been but totally rejected from a

[4]The linkage of the function has been handled differently depending on the dialect of LISP used. Our favorite solution has been to use the function cell of an interned atom of the same name as the node.

cognitive point of view. It requires that every member of a class is described by necessary and sufficient conditions.

The prototype view as developed by E. Rosch[5] describes a "prototype" as a summary description of all the members of a class. The third theory of categorization is the exemplar view. The exemplar view differs from prototype theory in the following way. The summary description used by prototype theory is not necessarily identical to any existing member of the category. Exemplar theory on the other hand postulates the use of one or more stored real exemplars of the category, in other words **no** summary description exists.

The exemplar view of categorization permits us to think in terms of upward inheritance from an individual to a class, because if we do not assume a summary description we may not associate attributes with it, and then the only source to derive inherited attributes from are other exemplars. This implies that it must be possible to inherit attributes from one exemplar upwards to a class and then back downwards to another exemplar.

For example, a knowledge base in our system might contain an object with no specified form that belongs to a class hierarchy. Classical downward inheritance would search up in the hierarchy until at some higher level a form is encountered. However, it might happen that no form is found anywhere in the hierarchy. In our interpretation of the exemplar theory it is valid to do a *"down"* search in the hierarchy for an object that belongs to the same class as the current focus object, and to inherit an existing form with *"up-and-down inheritance"* for it.

The idea of up-inheritance is not popular in AI. It is either ignored or explicitly prohibited. For instance, knowledge representation of the NETL style[6] which is based on marker passing, prohibits the idea of inheritance according to an up-and-down-movement because if one would permit markers to move up and down in the network the whole network would eventually be marked.

One is tempted to interpret up-and-down inheritance by considering the first step (the up-inheritance) as a form of generalization or inductive learning. However, this is not what we have in mind, because the representation of the class itself is *"not"* changed by a step of up-and-down inheritance. If a class should have many members only one of which has a form, and if this form should be changed after one application of up-and-down inheritance, then the second application of this inheritance rule will supply the new form, not the old form. Were we talking about a step of generalization, then the class would preserve the form after the first use of up-inheritance.

Are we then making a decision for the universal use of exemplar inheritance and against prototype theory? Clearly this is not our intention, because up-and-down inheritance is *"only"* used when no sufficient information is associated with the classes used for inheritance *i.e.*, when our version of a summary description fails. We do not eliminate the use of a summary description!

For practical purposes we have limited the use of up-and-down inheritance in two ways.

[5]E. Rosch. Principles of Categorization. In E. Rosch and B. Lloyd, Eds. *Cognition and Categorization.* pp. 27–48 (1978).

[6]S. E. Fahlman. NETL: A System for Representing and Using Real-world Knowledge. MIT Press (1979).

Up-search is done only from the lowest level, the level of the individuals, to the level immediately above it, *i.e.*, to the lowest level of classes. If there is no other member in this class, or if the other members do not carry the desired information, then up-and-down inheritance fails. One can argue that this does not make complete use of the class hierarchy, but it seems like a reasonable compromise, because humans use hierarchies that are flat and bushy. Rosenfeld has even argued that it is not necessary to view operations on hierarchies as recursive to an arbitrary depth, because this constitutes an unnecessary effort if one has only a flat hierarchy.

Secondly, up-and-down inheritance is used only for information that is urgently needed, and not as the default case. In a graphics system the one item of information that is obviously needed most is the form of an object, for which no "reasonable defaults" can be supplied. We will now formally define up-and-down inheritance which we also refer to as exemplar inheritance.

**Definition: Exemplar Inheritance.** If an individual is missing information about an important property, and this property cannot be derived by inheritance from a superclass of the individual, then the property may be inherited from any of the other members of the *immediate* superclass of the individual.

In our domain only "forms" are considered important, and we have therefore decided not to represent the fact that a property is important by an explicit assertion.

It is not yet clear what happens when several members of a class offer different properties for upward inheritance. In such a case a combined strategy of majority and recency may be used.

### Reasoning

The major reason for introducing the notion of graphical deep knowledge as separate from graphical knowledge has been the interest in doing reasoning about graphical structures. The first step of making a corpus of representations accessible to logic based reasoning is to transform it into a well formed declarative format with a defined syntax and semantics. It has been the approach of this investigation to limit the procedural representations which at some point are not avoidable in graphics to a small area, namely to iconic primitives. All conceptual relations between these iconic primitives are represented declaratively.

The second step is to formally define reasoning patterns. SNePS provides two different facilities for doing so, a system of rules and a system for defining paths. Although the rules that can be defined are very powerful and permit quantification as well as the use of non standard connectives we have chosen to concentrate in our implementation on the use of paths which are more efficient.

Path based inference in SNePS assumes that one has a node of a well specified category available (typically an "object") and follows the arcs that are pointing to this node backwards until one hits a node describing unknown and interesting information (for instance a "form" or one coordinate of a position). The well specified case frames of GDK assure

that if the required information exists at all in the network, then it will be reachable by a well defined path.

### Maintenance Interface

The use of the TINA program 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 an older version[7] 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 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. To our own surprise no integratory debugging was necessary!

The most complicated device that was "maintained" with the combined maintenance reasoner/ graphics interface has been a 6 channel PCM board. In Fig. 3.6.1 a screen dump from a GIGI terminal is shown. The PCM board does analog/digital coding and decoding, and its main components are inverters, transformers, and one PCM chip per channel. The large number of components and the limited quality of the involved hardware unfortunately resulted in a somewhat fuzzy diagram.

## 3.6.3 Graphics Interface for VMES on TI Explorer

Currently, VMES is running on the TI Explorer, and a Graphics interface for displaying the state of the system as diagnosis proceeds, is highly desirable. This will, in particular,

[7]Based on a VAX 11/780 and a GIGI graphics terminal.

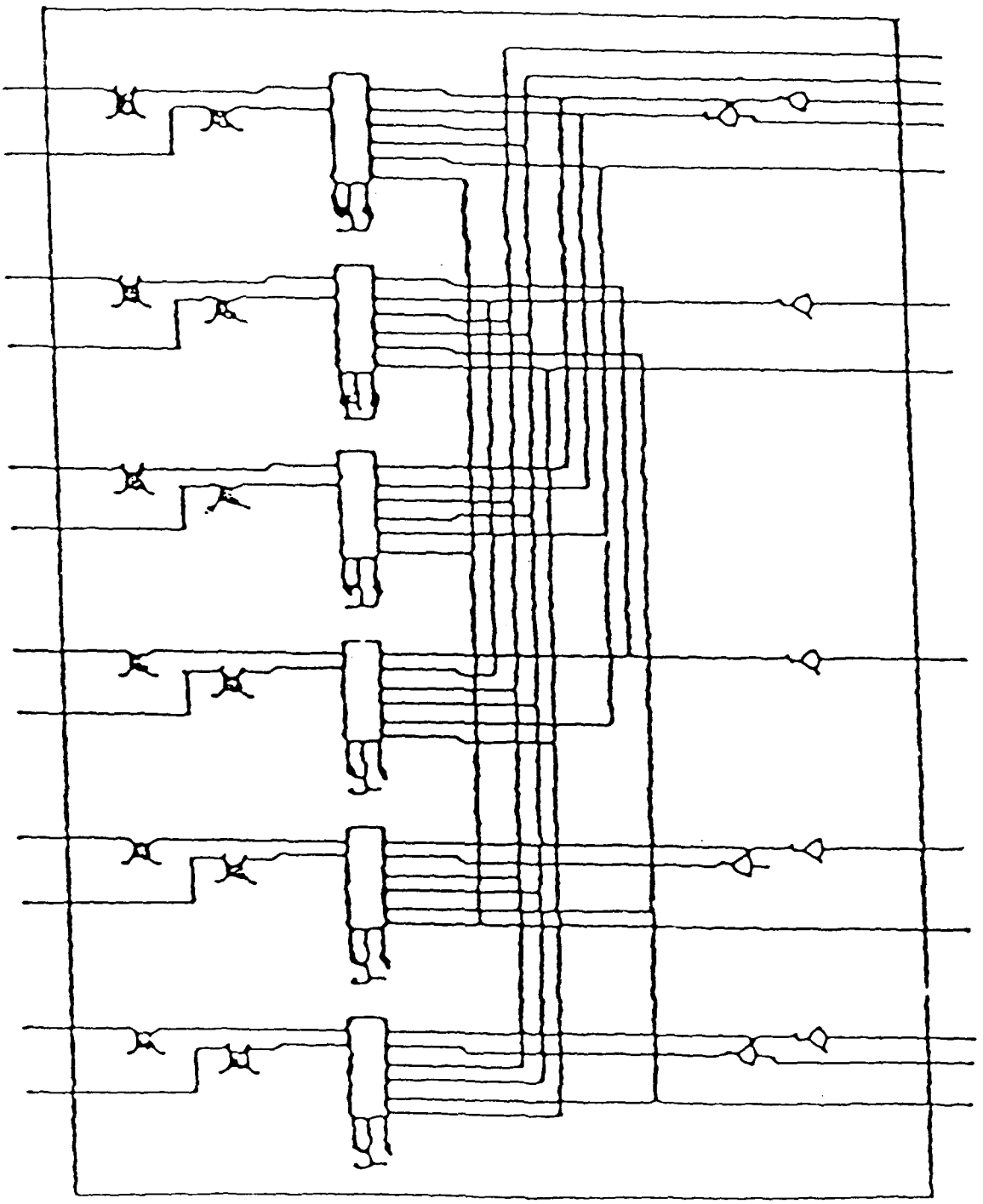

Figure 3.6.1: A screen dump of the PCM board

make the expert system more accessible to the maintenance technician who will not need to have an extensive knowledge of the VMES program or device representation scheme, to use the expert system effectively.

Work towards designing of such a Graphics interface on the TI Explorer has been started. Effective techniques for information transfer from the VMES knowledge database to the Graphics routines are being studied. When complete, the Graphics interface will display the circuit board under investigation. The suspect components, the component currently being diagnosed by the system, and the faulty components (as concluded by the system) will be identified using various color codes.

## 3.7 DEVELOPMENT OF SNEPS-2

The continuing development of the new version of the Semantic Network Processing System (SNePS-2) has proceeded through the year. During this past year, the mainstream of our system's development has been split in two directions: *(1)* the development and enhancement of the SNePS-2 system itself, and *(2)* making the system operational and maintainable over seven different computer systems.

The development of the SNePS-2 system has been focussed in three main areas: *(1)* the Natural Language Interface (NLINT), *(2)* the SNePS Inference Package (SNIP), and *(3)* the system's overall efficiency. The NLINT package, which consists of an augmented transition network (ATN) parser/generator and a lexicon, has been developed to a stage of competence that it is currently being used as the input system of choice for two other RADC sponsored programs here at SUNY at Buffalo. The SNIP package, still under development, has evolved to a point where both forward and backward node-based inference is operational over most of the system's possible types of rules including: or-entailment, and-entailment, numerical-entailment, and-or, and thresh. In addition to this, path-based inference, as part of the core functions of SNePS-2, is operational. Due to these developments, the VMES project has ceased using the now obsolete SNePS-79 system running on a VAX-11/785, and is now using SNePS-2 as its implementation vehicle on the Texas Instruments Explorer. With this much of the system up and running, we have started to perform profiling operations so that we can analyze the operation of SNePS-2's inference package and determine where we can improve its efficiency. Further, we have reorganized portions of the semantic network pattern matcher so that several data structures are no longer necessary and tens of functions have been removed from the code.

In keeping with our goal of developing a portable version of the SNePS system, we have stayed as close as possible to the definition of the language "Common Lisp", as described by Guy Steele. To help us with this task, we have taken it upon ourselves to get SNePS-2 running on as many different computer systems as we could, thus insuring portability not only within the language, but across architectures as well. The seven vehicles that currently run SNePS-2 are: Texas Instruments Explorer II, Symbolics 36xx, VAX-11/785, Sperry 7000, Encore Multimax, SUN-3/60, and Hewlett Packard 9000. The most difficult portion of this task has been overcoming the idiosyncracies of the Common Lisps running on some of these target machines. However, all but the HP implementation are currently being supported by essentially the same source code.

This year has seen SNePS-2 go from a half-implemented system, to one that is the knowledge representation vehicle of choice for three RADC sponsored programs being conducted here at SUNY at Buffalo.

## 3.8 VMES PUBLICATIONS IN 1988

Geller, J., "A Knowledge Representation Theory for Natural Language Graphics", Technical Report 85-15, Department of Computer Science, SUNY at Buffalo, Ph.D. dissertation.

Srihari, S. N., "Applications of Expert Systems in Engineering", chapter 1 of *Knowledge-Based System Diagnosis, Supervision and Control* by S. G. Tzafestas (ed), Plenum Press, 1988.

Chen, J. S., and Srihari, S. N., "A Method for Ordering Candidate Submodules in Fault Diagnosis", Technical Report TR-8736, Northeast Artificial Intelligence Consortium.

Martins, J. P. and Shapiro, S. C., "A Model for Belief Revision", *Artificial Intelligence*, 35:25–79, 1988.

## 3.9 TRIPS AND ACTIVITIES SUPPORTED BY RADC

### 3.9.1 Trips Funded

Executive Committee Meeting, Syracuse University, NY, January 25, 1988: Srihari.

SUNY at Buffalo, Department of CS, February 14–19, 1988, work with the VMES research group: Shapiro.

NAIC Spring Topical Meeting, Washington D.C., March 29–30, 1988: Srihari.

SUNY at Buffalo, Department of CS, April 11–15, 1988, work with the VMES research group: Shapiro.

26th Annual Meeting of the Association for Computational Linguistics SUNY at Buffalo, June 7–10, 1988: Shapiro.

NAIC Meeting with Rockwell International, Syracuse, NY, June 8, 1988: Srihari, Shapiro.

NAIC Executive Committee Meeting, Hartford/Springfield Bradley Airport, July 18, 1988: Srihari.

NAIC Annual Meeting at Minnowbrook Conference Center, Blue Mountain Lake, NY, August 8–11, 1988: Srihari, Upadhyaya, Chen.

NAIC Committee Meeting, Syracuse University, September 27, 1988: Shapiro.

Second Annual RADC Technology Fair, Utica, NY, November 15, 1988: Shapiro, Campbell, Chen.

### 3.9.2 Local Conferences Partially Supported

**Third Annual University Of Buffalo Graduate-Conference in Computer Science (UBGCCS-88)**

David Satnik and Lynda Spahr were on the organizing committee for this conference, held at UB on March 15, 1988. Six speakers from SUNYAB and two each from the Universities of Rochester, Waterloo, and Toronto participated. Scott Campbell, with Paul Palumbo, edited Tech Report 88-03, "UBGCCS-88 Proceedings of the Third Annual UB Graduate-Conference of Computer Science".

## Twenty-sixth Annual Meeting of the Association for Computational Linguistics (ACL-88)

The Twenty-sixth Annual Meeting of the Association for Computational Linguistics will be held on the SUNY at Buffalo North Campus, June 7-10, 1988. The local arrangements were co-chaired by Dr. William J. Rapaport and by Lynda Spahr, secretary at UB for NAIC. Other RADC sponsored volunteers included Jiah-shing Chen, Scott S. Campbell, David Satnik, Deepak Kumar and Syed Ali.

Approximately 330 linguists and artificial-intelligence researchers from universities and industry in the U.S., Canada, and overseas attended this conference, which offered an unparalleled opportunity for our faculty and graduate students.

This was also the most successful ACL conference in terms of external support, with IBM, DEC, Barrister, Calspan, and CUBRC donating a total of $4700 towards the conference. Other financial support came from the SUNY Buffalo Conferences in the Disciplines, FNSM, FSS, the Graduate Group in Cognitive Science, and the GRI in Cognitive and Linguistic Sciences.

# *MISSION*
# *of*
# *Rome Air Development Center*

*RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence ($C^3I$) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of $C^3I$ systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.*