Military Technical College
CAIRO - EGYPT

AN EXPERT SYSTEM TO DISCOVER CAUSE OF FAILURE FOR IBM MICROCOMPUTER PERIPHERALS INTERFACE

Afaf M. Albetar * and Dr. Nabil A. Ismail **, Senior Member IEEE

ABSTRACT

A knowledge-based system to discover cause of failure for the IBM microcomputer peripherals interface has been built using a backward-chaining inferencing expert system framework. A detailed technical reference manuals are currently used for peripherals interfacing maintenace and failure diagnosis. Task complexity limits this method. A set of real-time self-test routines, database, explanation facility and interactive menus have been built to compensate for the inadequate man-machine interfaces to this system. The resulting diagnosis system, we call it MUES (Menoufia University Expert System) incorporates: (i) databases to store data, control and status registers contents, I/O ports configuration modes and communication format, (ii) interactive menus, and (iii) an intelligent rule-based system which takes its evidence in real-time via a set of assembly modules and provides advice on the cause of failure to the operator. MUES written in Turbo Prolog and running on an IBM XT microcomputer, uses if-then production rules to encode the knowledge and is based on the backward-chaining inference engine that allows rules to look very much like English. The interaction between the operator and the knowledge-based diagnosis system has been reduced to a small number of questions requiring yes/no answers. The explanation facility requires how/why questions. The remaining input to the diagnosis system is provided by status evidence (logic values low/high) as a result of executing a set of self test routines written in assembly language.

---

* Graduate student, ** Associate Professor, Dept. of Computer Engineering & Science, Faculty of Electronic Engineering, Menoufia University, Menouf 32953, Egypt.

PROBLEM DEFINITION

Discovering the cause of failure for a computer-peripheral interface is an evolutionary process consisting of a series of trial and error operations. A detailed technical reference manuals are currently the only source available for solving the problem. Task complexity limits this method. In this paper we will discuss a case study knowledge-based system program suitable for discovering the cause of failure for the IBM PC/XT/AT microcomputer-peripherals interfaces.

A knowledge-based system [1],[2] can be described as software program that uses a series of emperical values about a topic coupled with an inference procedure, to draw conclusions from an input set of facts. Expert systems (ES) are specific type of intelligent knowledge-based systems concerned with problem solving applied to a particular domain, and exhibiting a high performance in decision making.

Real world expertise must be structured in a useful form, stored, and efficiently accessed. Many expert systems [3],[4] employ the production rule representation scheme [5] because it seems natural and easy to express in IF-THEN statements rules of thumb, hypotheses based on some premises, causality, and so forth. Rule based expert systems require at least a Rule Database (RDB) of production (condition-action) rules held in production memory, as well as a database of currently true assertions, typically referred to as the Working Memory (WM). In addition, some expert systems utilize a large Fact Database describing the problem domain at hand, this database may be separate or may be combined with either the Rule Database or Working Memory, depending upon system architecture, e.g. the R1/XCON system [4] and ACE system [6].

The production rules found in expert systems describe consequences of certain information, and typically in the form that if certain conditions are true then take certain actions:

$$P_i\ (C_1\ C_2\ \ldots\ C_n \longrightarrow A_1\ A_2\ \ldots\ A_m)$$

Conditions are forms that are instantiated by memory elements. Actions add elements to working memory or modify existing elements or provides advice to the operator. The working memory can be accessed by all the rules, no part of it is local to any of them in particular. Rules do not call other rules; communication between rules occures only through the working memory.

There are two forms of reasoning with production rules: forward-chaining and backward-chaining [5]. Forward-chaining (also known as data-driven or event-driven) works from available evidence toward conclusions. In other words, it looks at the conditions of a rule and if they are confirmed by the evidance at hand, the actions of the rule are performed. The system therefore is continually scanning its rules and attempting to match their conditions with current facts in order to make new inferences. It stops when it finds the desired solutions or when there are no applicable rules.

Backward-chaining (goal-driven or expectation-driven), on the other hand, starts with a hypothesized conclusion and works backward towards the supporting evidence. According to this strategy, the system looks at the action part of the rules and finds one that would conclude the current goal. It then takes the conditions of the applicable rule and seeks other rules whose actions would satisfy these conditions. The process repeats itself until all

goals match the known facts or until all applicable rules are exhausted. In general, a backward-chaining rule has the form

conclusion:-premises.

The system discussed in this paper has been using a backward chaining.

SYSTEM DEFINING CHARACTERISTICS

The system under investigation, we call it MUES (Menoufia University Expert System) is a rule-based diagnosis system that has much in common with other domain-specific systems that have been published in the literatures over the past several years [3],[4]. MUES's domain of expertise is discovering the cause of failure for the IBM PC/XT/AT microcomputer/peripherals interfaces. IBM microcomputers are the most powerful micros that have been used in the University for teaching and experimental labs. Its architecture have been fairly understood by a wide spectrum of specialist staff and students at the University. The IBM microcomputer uses an appropriate high speed bus interfaces that can support a wide variety of peripheral devices [7], [8]. A typical system contains not very little number of components include peripheral devices, drivers for the devices and cables. There are a large number of rules that dominate the way in which these components work, and hence the causes of failure may be discovered. The understanding of the IBM microcomputer/peripheral interface will lead to diagnostic of possible failure, this failure may be partially or full distructive to interface. Both cases must be considered, since a partialy fault will make the system respond to some activation signals (messages). Fully fault will prevent the system to respond to any signal, this will prevent the system from using the aid of these signals. Completeing the hardware diagnosing stage leads to the stage of knowledge abstraction in which we will try to represent the human expert knowledge in a knowledge model that is acceptable by the computer.

MUES is implemented in Turbo Prolog, which is developed by Borland International [9], linked with assembly language routines. Turbo Prolog supports the Assembler calling convention by using its associated extension in the global declarations. It has an efficient deductive reasoning capabilities and uses a backtracking mechanism [10] which, once one solution has been found, causes Turbo Prolog to reevaluate any assumptions made to see if some new variable values will provide new solutions.

Production memory contains all of the MUES's permanent knowledge about how to discover cause of failure for computer peripheral interface and provides advice on the cause of failure to the operator. MUES currently has 530 rules that enable it to perform the task. These rules cover the parallel printer, RS232, keyboard, VDU display, and disk drives interfaces. MUES's database, which may be filled by the user through user/system dialog or, as in our case, by using a simple test routines which will provide us with a peripheral status. These routines have been written in assembly code, they mainly acquire the status of a peripheral or send an activation signal or message to the peripheral.

SUPPORT ENVIRONMENT

Explanation Facilities

An expert system that is capable of explaining its own behaviour and reaso-

ning processes generally improves users' acceptance. We have adopted two basic types of explanation; why and how. The first is characterized by the question:

"why are you asking me such question?".

and the second by the question:

"how did you come up with this conclusion?".

The why-type question can be ambiguous at times for it may actually demand a sequence of justifications leading to the question, in which case there is no real distinction between why and how.

End-user/system interfaces

The end-user/system interfaces include menu selection, simple question-answering, short English-phrase input, and user/operating-system communication. Menu selection is a very popular end-user/system interface in which a user sees a menu and is asked to make a selection from the displayed options. A simple question-answering interface is very handy when all we need from the user is a simple yes-or-no response, or a keyword limited to a given range of numbers or characters.

System/Outside-system interface

Interactions between an expert system and the outside world are absolutely necessary for many real-world applications. For example, it is important to be able to re-use well-tested interface routines written in Assembly language, and to have the control accessing an external databases. In light of this the system enables a knowledge engineer to invoke an external procedure within a Turbo Prolog rule. Argument passing (status flags, register contents and control ports setup) is done via input and output files allocated exclusively for this purpose.

Debug facilities

Turbo Prolog provides a very powerful debug package to spy procedures, trace execution, and step through programs. The debug facility describes the execution of a Turbo Prolog programs in terms of CALL, EXIT, REDO and FAIL events [9].

EXAMPLE RULES

Parallel Printer Interfacing Example Rules

Figures (1)-(4) depict the connector, control output signals, status input signals, and the parallel printer interface timing diagram respectively. This information has been transferred into IF-THEN production rules for our diagnosis expert system. The following shows examples of these rules written in Turbo Prolog.

In this rule the argument of the subgoal 'is' represents the status evidence (logic values Low/High) which correspond to the contents of the status and control ports in the database.