ARBY: Diagnosis with Shallow Causal Models

Drew McDermott
Yale University Department of Computer Science

Ruven Brooks
ITT Integrated Systems Center

ABSTRACT

Arby is a software system or higher order language for writing expert systems to do diagnosis in electronic systems.

As such, it is similar to EMYCIN (Van Melle 1982) in application, but quite different in design. It is rule-based to an extent, but the rules are written in predicate calculus. It resembles Caduceus (Pople 1977) in its mechanisms for refining and combining hypotheses.

1. Overview

Arby is a software system or higher order language for writing expert systems to do diagnosis. As such, it is similar to EMYCIN (Van Melle 1982) in application, but quite different in design. It is rule-based to an extent, but the rules are written in predicate calculus. It resembles Caduceus (Pople 1977) in its mechanisms for refining and combining hypotheses.

Unlike both of those systems, however, Arby is designed for finding faulty modules in large electronic systems. (Davis 1981) The system we have been studying so far is the Microwave Stimulus Interface (designed by the General Dynamics Electronics Division). This system is itself used to test avionics equipment. It provides microwave signals modulated in certain standard ways. A (simplified) block diagram appears in Figure 1.

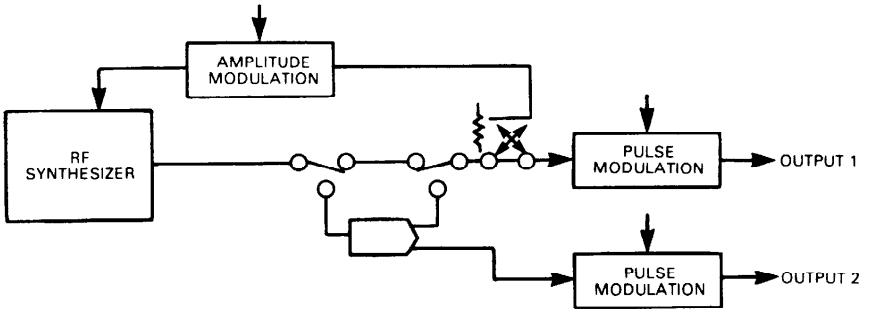

Figure 1. MSI block diagram

The MSI consists of two main stages. The first amplitude-modulates the output of an RF synthesizer using a feedback through a detector. The second introduces pulse modulation. Ordinarily, the diagnostician is presented with a faulty MSI which has already been run through some automatic diagnosis software. So he is presented with *findings* and *evidence* including the output of the low-level diagnostics. His job is to find which module (detector, pulse modulator, any of several switches, etc.) is faulty. He can set switches to any position, probe with an oscilloscope, break the connection at certain points, and stimulate various inputs with standard signals. However, these actions vary in time expended and information gained.

This domain is distinguished by the following two features:

- Reasoning with shallow models

  The problem of diagnosing faults in electronic circuits is in general quite difficult. (deKleer 1976, Brown 1975) This is because electronic circuits are hard to break into separable modules. Explaining why such a circuit is malfunctioning requires understanding a causal model of it. However, above the level of individual circuits, the models get simpler. They usually consist of a flow of something (like a signal) through *nodes* and *devices*. The most sophisticated system to understand is a negative-feedback loop. On the other hand, things are more complex than in medical diagnosis, where multilevel causal models are quite rare.

- Information acquisition strategies

In diagnosis, the full set of symptoms is not usually available at the time diagnostic reasoning is done but must be acquired during the reasoning process; thus, part of the reasoning process must be to determined which tests are to be run or which observations are to be made. The mechanism for doing this must take into account the cost - whether financial, time or risk to the patient - of acquiring the additional information.

2. How Arby Works

Arby consists of two major modules, an inference system, HYPO, which reasons about symptoms and hypotheses and a human interface system, IFM, which obtains information from the user. Both modules of Arby are written in Franz Lisp, and use the DUCK general purpose retriever. DUCK is a descendant of Conniver (Sussman and McDermott 1972), and may be thought of as a more powerful version of Prolog.

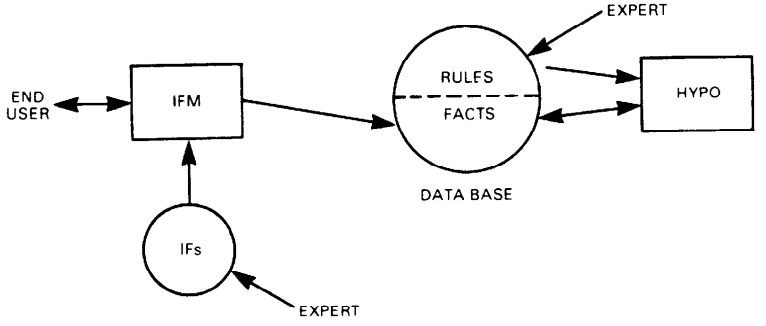

Figure 2. Arby block diagram

2.1 The Hypothesis Generator

The hypothesis generator (HYPO) is responsible for creating and refining hypotheses to account for findings. It has two main submodules for doing this: the candidate hypothesis finder, and the evidence sifter. Each of these relies on predicate-calculus deductions on a data base of rules.

Findings and hypotheses are (in principle) arbitrary formulas about entities in the domain. A finding is a fact that needs to be explained, and a hypothesis is a fact that would explain it. In order to find hypotheses, it uses the backward-chaining facility of DUCK to find solutions to the goal:

(accounts-for ?hypo <finding> <given>)

(The third argument will be explained later.) The solutions come back in the form of possible values for ?hypo, plus justifications, rules and facts which together support this explanation. (Doyle 1980)

It is quite common for HYPO to find more than one hypothesis that could account for a given finding. To handle this, it enters a special *choice protocol* (McDermott 1978) in which it attempts to find evidence for and against each candidate. That is, it attempts to deduce formulas of the form

(evidence <hypothesis> ?amt)

for each candidate hypothesis. It totals the evidence, and the candidates with totals significantly higher than the others are retained.

This strategy as described has one failing: it assumes that all the evidence has already been gathered. In some domains (e.g., that of MYCIN), this is a reasonable assumption. In the domain of simple electronic reasoning, there are many tests that could be run. It would be silly to run them all before the consultation begins. Instead, the system doesn't run any until it is faced with a choice situation that requires more information to separate the two leading candidates. Then it runs whatever test seems most profitable, that is, the one that will apparently produce the biggest impact on the evidence totals for the leading candidates at the least cost. (This is done with the Interaction Frame Manager, described below.)

If no test looks like it is worth running, then HYPO is not lost. In addition to favoring hypotheses with the highest evidence totals, it also favors those that account for as many findings as possible. (Pople 1977)

After producing the composite hypothesis with the best overall evidence, Arby does not quit, but proceeds to *refine* each element of the hypothesis. This means finding a more detailed explanation of each finding. For instance, the first round might localize the failure in module33. The second round might then localize it to a device within that module, by solving the deductive goal:

(accounts-for ?hypo <finding> (fault-in module33))

The advantage of doing things this way is that only a small number of alternatives need to be considered at each level. When one is rejected, all of its refinements are swept away without ever being considered. Furthermore, this organization nicely reflects the hierarchical structure of many electronic systems.

2.2 Interaction Frames

The basic structure which controls
the user interaction is organized is an interaction manager, IM, and a set of interaction frames, IFs. An IF is a discrete unit of interaction with the user which, when it terminates, results in the addition of assertions to the database. A very simple IF might consist of asking the user a multiple choice question and, depending on the answer, placing just a single assertion into the database. A more complex IF might instruct the user step by step to run some complicated equipment test and then place several different assertions into the database to record the results of the test. An important property of interaction frames is that the assertions they may add are unconstrained; this is in contrast to the MYCIN structure in which an interaction may only change the values of variables.

Interaction frames are invoked because the HYPO component needs to retrieve information that is not yet in the database but which could be obtained by asking the user. This is implemented by asserting rules of the form:

(<- <form to be deduced>
  (**QIF** <interaction frame> <arguments>))

If the deduction can not be made any other way, this causes the interaction frame to be placed on a list of candidate frames to be invoked. Using the cost criterion described earlier, this list is ordered to identify the most "profitable" frame to run.

In general, however, just asking for the information in the order determined by the inference system will not result in an acceptable order of questions from the user's standpoint: The questions might have logical prerequisites which are not yet satisfied or the questions might not occur in the order to which the user is accustomed. To insure the satisfaction of these constraints, the Interaction Frame manager requires that an IF be explicitly enabled before it can be invoked. In the database, this is indicated by assertions of the form:

(want <IF> <arguments>)

To insure, for example, that the status of switch 17 is asked about only after it has been determined that the level of amplitude modulation is high, rules of the following form could be written:

(<- (want IF23 17) (modulation-level high))

There are also assertions for indicating that an IF has already been run and for indicating that an IF may be run whenever needed. This combination gives the Interaction Manager considerable power in reasoning about and optimizing the structure of interactions with the users.

2.3 The Arby Environment

A system like Arby is, we have discovered, one-third thinker and two-thirds explainer of its thoughts. Although the set of explanation tools is still rapidly evolving, we have found it necessary to provide tools for the following tasks:

- Editing of rules and IFs by the domain expert.
- Explanations of why a question is being asked (in terms of the evidence it might provide)
- Explanations of why a given fact is currently believed.

3. Example

Version 0 of Arby has been written, and coding is complete for an intial set of rules and interaction frames for the MSI. We anticipate the usual redesign as we attempt assimilation of expert knowledge, but the basic design seems sound.

As an example of the operation of Arby, we present the following description of current system functioning:

1. When Arby is started, a standard IF is always run to ask what the output of the low-level diagnostics was. Let's say the output was "persistent loss of power at low frequencies."
2. Arby notes this as a finding to be accounted for.
3. "Accounts-for" rules are used to propose candidate explanations. In this case, the problem could be in the low-frequency modulation loop. This is the only candidate, so it is accepted.
4. Next Arby attempts to refine the hypothesis. "Accounts-for" rules suggest any of several components in the low-frequency modulation loop. "Evidence" rules are sought to propose evidence one way or another. There is a rule of the form

   "If after cutting the loop before the synthesizer,
   measurement of a node shows it to be saturated
   the wrong way,
   then
   there is good evidence in favor of
   localizing the problem in some device
   before the node
   else
   there is good evidence in favor of
   localizing the problem in some device
   after the node."
5. This rule has no immediate effect, because it refers to a real-world action that may not be worth performing. This action is queued, and, assuming no decision can be reached without it, it is handed to IFM. IFM then uses IFs to instruct the user how to cut the loop and what to measure.

References

[1] Johan deKleer 1976 Local methods for localizing faults in electronic circuits. *MIT A.I. Lab TR394*

[2] Jon Doyle 1980 Truth Maintenance in *Artificial Intelligence 12* pp. 231-272.

[3] Allen Brown 1975 Analytical knowledge, causal reasoning and the localization of failures *MIT A.I. Lab. TR394.*

[4] Randall Davis 1981 Expert systems -- where are we and where do we go from here? Invited talk at *IJCAI 7*

[5] Drew McDermott 1978 Planning and acting. *Cognitive Science 2*, no. 2, p. 71

[6] H. Pople 1977 The Formation of Composite Hypotheses in Diagnostic Problem Solving, *IJCAI 5.*

[7] Gerald J. Sussman and Drew McDermott 1972 From PLANNER to Conniver -- a genetic approach. *Proc. FJCC 41*, p. 1171

[8] Van Melle, W. 1980 A domain independent system that aids in the construction of knowledge based consultation programs. *Heuristic Programming Project, Department of Computer Science, Stanford University, HPP 80-11.*