11. Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II and III

JOHN SEELY BROWN, RICHARD R. BURTON AND JOHAN DE KLEER
Xerox Palo Alto Research Center, Coyote Hill Rd, Palo Alto, California, CA 94304, U.S.A.

This paper provides an overview of the pedagogical and knowledge-engineering techniques in SOPHIE I, II and III. From the pedagogical perspective, we focus on the kinds of interactive instructional scenarios that have been made possible by integrating explicit models of domain expertise into the instructional medium itself. We provide both concrete examples and discussions of their significance. From the knowledge engineering perspective, we focus on a set of abstractions and techniques that have enabled us to build intelligent systems that are efficient, habitable and exceptionally robust.

1. Introduction

1.1 HISTORY AND OVERVIEW

The research described in this paper took place over a five-year period and centered around three different SOPHIE systems. Work began on what was to become SOPHIE I in early 1973 when two of us, Brown and Burton, were at the University of California at Irvine. The Air Force had expressed interest in using computers in their advanced electronic-troubleshooting course, particularly in the laboratory section; and we were interested in exploring interactive learning environments that encouraged explicit development of hypotheses during problem-solving by facilitating the communication of the student's ideas to the machine and by enabling the machine to critique them. We recognized electronic troubleshooting as a particularly good application domain. While at Irvine, the general-purpose electronic simulator, SPICE, (Nagel and Pederson 1973) was obtained, an appropriate electronic device was chosen for study (the IP-28 regulated power supply), and protocols were collected of experts troubleshooting the IP-28.

In the summer of 1973, the project moved to Bolt, Beranek and Newman, Inc. in Cambridge, Massachusetts. The next year saw the development of a SPICE model of the IP-28 which was capable of being faulted, a fledgling natural language interface, a semantic network database of information about the IP-28, and implementation of the basic pedagogical and inferential ideas. By the end of 1974, a version of SOPHIE I was being used in exploratory experiments [Brown, Burton and Bell 1974, 1975]. During the following year, the habitability and robustness of the system were improved through continuous usage. Our major addition during this time was a dialogue mechanism that allowed the natural-language interface to handle a large number of elliptic and anaphoric expressions that arise in a troubleshooting context. By early 1975, SOPHIE I was essentially completed [Brown and Burton 1975].

Most of SOPHIE I's intelligence resided in a collection of procedural specialists that judiciously selected, set up and ran "critical" experiments on a general-purpose circuit simulator. This allowed SOPHIE I to evaluate a student's hypotheses, critique his measurements and handle almost any question presented in the context of electronic toubleshooting.

SOPHIE II, built the following year, was an extension of SOPHIE I designed to be self sufficient for field testing [described in Brown, Rubinstein and Burton, 1976]. This required additional pedagogical components as well as supporting course material. Of special pedagogical interest is its expert troubleshooter, which enabled students to insert arbitrary faults into the circuit and watch the expert locate them. The key pedagogical benefit comes from the expert explaining both its tactics for choosing measurements and, more importantly, its higher level strategies for attacking the problem. In this manner, the student is not only exposed to the general strategic principles of problem-solving, but also sees them applied to a problem of his own making. The expert uses decision trees to guide its measurements and contributes little to the technology of building intelligent systems. It did, however, enable us to investigate the structure of explanations for flexible, intelligent systems such as SOPHIE III.

SOPHIE III was developed over a two year period starting in 1976. There were a variety of motivations for it. First, the experiments run with SOPHIE II pointed out additional capabities needed by a coach or automated lab instructor. Our ability to implement these capabities required (a) a more powerful and human-like reasoning engine, (b) techniques for modelling the student and (c) coaching strategies that limited the coach's interruptions of the student and governed its suggestions when it did break in. This latter problem also required understanding how explanations could be synthesized from the collection of non-uniform inference strategies we expected to use. Exploring all these issues simultaneously on a task domain that stretched our ability just to build the expert, seemed unwise. Thus, in the SOPHIE III we concentrated on building an expert electronics reasoner [de Kleer 1976, Brown 1977] that had some of the characteristics necessary to facilitate the student modelling and coaching subsystems. Concurrently, we chose to explore the more complex aspects of the coaching and student modelling components on the considerably simpler task domains of games [Burton and Brown 1981] and elementary mathematics [Brown and Burton 1978, Burton 1981], domains for which the building of a reasoning engine could easily be done.

In addition to our desire to study issues of explanation, the SOPHIE III project was started for the second reason of exploring the tradeoffs between generality and efficiency in building intelligent systems. SOPHIE III sought to mix very general knowledge of electronics with very specific device-dependent information hopefully resulting in a system both general and efficient.

The third reason for building SOPHIE III was to explore expanding the role of the coach to be more like that of an on-the-job consultant. In particular, we wanted a system that could track and advise the student as he was working on a

real piece of equipment, not just on one being simulated in the computer. To this end, we wanted to understand more about how to merge intelligent job performance aids with on-site job training.

1.2 PEDAGOGICAL OVERVIEW

Perhaps a comment is in order concerning the kind of troubleshooting expertise that we seek to develop in users of the SOPHIE environment. Many troubleshooters typically have great difficulty fixing familiar devices with unusual faults or unfamiliar devices for which they have had no specific training. This is *not* the kind of troubleshooter we wish to develop. Instead, we focus on producing a skilled troubleshooter with a sufficiently good conceptual — albeit qualitative — understanding of electronics to be able to develop appropriate diagnostic steps on his own, to digest new information from technical manuals and to troubleshoot unfamiliar equipment.

However, much more than the art of troubleshooting can be learned by studying the troubleshooting process. People who are skilled at operating complex equipment often exhibit proficiency at handling the unusual situations or casualities that inevitably arise. Their ability to "patch" their routine procedures or to back out of a weird state they accidently produce, often requires the kind of common-sense, causal understanding that we seek to instill in our expert troubleshooters.

1.2.1 Reactive Learning Environments: A Philosophy

The educational paradigm developed in the SOPHIE systems focuses on experiential learning which capitalizes on episodic memory for encoding instantiations and ramifications of generic knowledge. When engaged in laboratory activities, students are forced to apply their factual knowledge to solve problems. The problem-solving process gives rise to experiences that structure the factual knowledge. To facilitate this style of learning, the student must be encouraged to formulate, test and witness the consequences of his own ideas and must be freed from worry about possible catastrophic consequences. Because the student's ideas have arisen while attempting to solve a problem, they are anchored to a personally meaningful context; it is within this context that the idea is best explored with the student. The system should be able to critique the student's ideas. In addition, the system should watch for situations in which, while working on one problem, the student produces an effect that illustrates a principle he has just learned. If he fails to perceive this connection, the system should call his attention to it, providing him with the choice of pursuing it.

Supporting metaphors for this style of learning environment stem from coaching, apprenticeship and laboratory instruction, in which technology can be used in three ways. First, it can make experimentation *easier* for the student. Second, technology can make experimentation *safer*. Third, technology can be used to help students *learn from their mistakes*. All three benefits of technology have been realized in our project, but the emphasis has been on helping students learn from their mistakes, especially as it applies to automating the role of a coach or lab instructor.

1.2.2 Kinds of Knowledge and Activities to Impart It

There are many different kinds of knowledge that need to be interwoven in order to establish a sound framework for expert troubleshooting. Troubleshooting *strategies* are responsible for controlling the sequence of measurements to be made. A good example is the "half split" strategy which directs the troubleshooter to choose his next measurement so that its result will halve the current set of hypotheses about what is at fault (assuming all are equally likely). Closely allied to strategies are the troubleshooting *tactics* which concern the ease of making one class of measurements over another. For example, voltage measurements are usually easier to make than current measurements and, unless the device is already disassembled, external measurements are easier to make than internal ones.

In addition, the good troubleshooter must have sufficent *understanding of electronic laws, of circuit components and of the overall functional organization of the device* to enable him to draw conclusions from his measurements. For example, he must be able to deduce whether a transistor is faulty from observations of its input/output behavior. He must know simple electronic laws (i.e. Ohm's and Kirchoff's) in order to propagate known measurements into new areas of the circuit. And to understand the more global consequences of his current set of measurements, he must have an understanding of the causality of underlying *circuit mechanisms* — of how negative feedback works, for example, or of how one component faulting might cause another to fail.

2. Pedagogical Techniques

In this section we discuss the various kinds of pedagogical environments that have been developed and explored with SOPHIE I and II. By so doing we hope to provide a sense of the educational challenges that must be faced in moving from an AI-oriented kind of research to one more concerned with cognitive and user-related issues. This section shows how SOPHIE's components form an integrated system and describes some of the pedagogical techniques being investigated. We focus on the SOPHIE II system because it is an extension of, and includes all of the features of, SOPHIE I.

2.1 SOPHIE II

The SOPHIE II system is a collection of components, each designed to foster activities that present to the student (or facilitate his discovery of) the kinds of knowledge needed for expert troubleshooting in electronics. These components consist of: 1) a "canned" articulate expert troubleshooter; a simulated laboratory (SOPHIE I); 2) a limited automated laboratory instructor or coach (SOPHIE I); 3) a variety of computer-based games; and 4) a collection of written material describing how to understand the function of a particular piece of equipment.

2.1.1 Articulate Expert Troubleshooter

The beginning student's first activity with the SOPHIE system is watching the articulate expert troubleshooter locate a fault. The articulate expert troubleshooter was designed to provide students with a nonthreatening, graceful introduction to the capabilites of the overall SOPHIE system and to expose them to expert strategic

and tactical reasoning. To enhance the student's interest the student is allowed to choose which function block of the device is to be faulted and the symptoms it is to manifest. Once the expert succeeds in locating the faulty function block, the student is given the chance to locate the particular faulty component within that block.

Unlike many AI systems the articulate expert foregoes the goal of general applicability to attain cogent explanations, especially pertaining to strategic issues. The underlying mechanisms of the expert are based on decision trees annotated with schema for producing explanations about troubleshooting a particular circuit. Since the annotations are prestored and constructed by hand, they can be made as insightful and structured as desired. Futhermore, since the annotations are associated with nodes in a decision tree and since any node can be reached by only one path, their exact context is known ahead of time. Each node has several schema with the choice depending, primarily, on how often the student has activated this node.

2.1.1.1 Brief Description of the Interaction

The interaction proceeds as follows: The "expert" first explains the procedure of the lesson — that it is going to try to isolate a faulted functional block selected by the student. Since the student will know what is wrong with the instrument (at a block level), he will be asked to predict qualitatively the results of measurements at the time they are made by the expert.

Next, the "referee," a program responsible for inserting student-specified faults and verifying student predictions, asks which block is to be faulted. The student may select any of the seven functional blocks in the IP-28, the device being debugged. Depending on the selection, the referee may ask for more specific information about how the external behavior of the block should be affected by the fault. The student knows only the selected external behavior, not the actual component fault, and will have to make predictions based only on this external behavior. The referee only installs faults which produce clear qualitative symptoms; the student should be able to propagate the symptoms through the device.

Next, the expert attempts to locate the fault, explaining its strategy as it goes. Before the expert makes each measurement, the student is asked to predict the qualitative behavior of the instrument (will the measured value be too high, too low or about right?). Depending on the particular approach taken by the expert, it may ask the student a question or two about what can be determined from the measurements made up to that point and comment on his response.

If the student makes a wrong prediction, the referee takes over to demonstrate the actual behavior of the faulted instrument. It does this by posing the laboratory module of SOPHIE a question, chosen to demonstrate the real behavior. The SOPHIE lab responds with an actual measurement, and the referee restates the measurement qualitatively, in terms of the expert's question. If the student doesn't know the answer to one of the expert's questions, he may ask for help. The referee will ask SOPHIE, as above, and summarize the answer.

When the expert has decided what block contains the fault, it announces this to the student, along with the final bit of reasoning which led to the conclusion. The student is asked whether the conclusion is correct and whether he would like to isolate the actual component fault. If he would, a summary of the expert's measurements (expressed as answers to SOPHIE lab inquiries) is printed, and the student is put into a SOPHIE lab with the faulted device. A history list of the measurements made by the expert during this session is provided so that the student may review them and their associated values. He thereafter may make any measurements he wishes, propose hypotheses, replace components and so on, as in a normal lab session.

2.1.1.2 Educational Rationale

There are several pedagogical reasons for conducting a dialogue between the student and the expert. The first is to convey a useful debugging strategy to the student. The expert views the device in terms of discrete boxes which interact in known ways. It has, in other words, a high-level causal and teleological model of how the circuit works.† Without having to make measurements inside functional blocks themselves, the expert is able to determine which block contains the fault.

A second reason for providing the expert debugging interaction is that it allows students practice envisioning the consequences of symptoms and faults, a skill that taxes their causal understanding of electronics in general and the circuit in particular. It also helps students become better at checking their own hypotheses. A final motivation for using the expert is to present a gentle introduction to the SOPHIE lab. When the student makes an erroneous prediction, he witnesses the referee using the SOPHIE lab to determine the right answer. A beginning student is thereby exposed to a variety of acceptable questions and wordings for later use during his own SOPHIE lab interaction. As a result, when he is given the opportunity to isolate the actual component fault within the faulted block, he has a good idea of how to use the lab without needing explicit instruction.

2.1.1.3 Evolving Models

One way of viewing the process of learning how to debug a device is to think of the student as moving from static to dynamic models. For example, in the case at hand, his initial understanding of the power supply might be summarized by the simple statement of intent: "The supply puts out a fixed voltage." This model contains no variables or internal constructs and is thus essentially static. A more refined model contains provision for one contingency: "The supply puts out a fixed voltage, unless its current limit is exceeded, in which case it puts out less." The model develops to encompass internal parts and more interactions: "The power supply puts out a voltage equal to the reference voltage, unless. . . ." and so on. More and more variables are added as the understanding of the power supply improves.

Mechanisms are required for pointing out to students the mismatches between their developing models and the real-world situation. Conflicts of model with fact

†When we speak of a teleological model, we refer to a model in which intended purposes are described.

are the raw material from which better models are made. The expert debugging interaction provides many small opportunities of this sort for students to improve their understanding. Every time a student makes a wrong prediction, he has an opportunity to go through a "What? That can't be! Aha!" cycle which improves the accuracy of his world view.

In order for such conflicts of model and fact to be used, they must be perceived by the student. To extend this notion, we chose to alternate lab and expert activities. Either alone is less effective than the combination. This is because troubleshooting provides not just motivation but periodically a set of seemingly contradictory observations. When the same contradictory observations are subsequently made by the expert, the student pays heightened attention to its method of successful resolution. Often the escapes from seeming-dilemmas focus the student's attention on details or complexities previously overlooked. Conversely, things which concern the expert — reasons for inference, caveats, and the like — may not be assimilated by the student on first sight but may make sense later in a debugging problem which he can solve by appealing to them.

2.1.1.4 Expert Debugging Strategies

The debugging tree used to implement the expert, as we said, incorporates only one of the many overall approaches which might have been employed. Its important characteristics are that it causes the expert to operate at a function-block level, to rely on qualitative measurements, to utilize multiple substrategies and to make measurements which are teleologically significant.

There are several reasons for operating at a function-block level. Most inexperienced troubleshooters tend to begin their testing at a very local level. In a circuit of even moderate complexity, such an approach will usually lead to wasted time and components. A function-block model of the instrument is a convenient mental shorthand for grouping collections of components, thereby enabling a higher-level approach and simplifying behavioral predictions about sections of the circuit.

The expert's debugging strategies rely only upon qualitative measurements. The sort of causal reasoning promoted by this qualitative approach develops the student's tendency to think logically. Chains of the sort, "Well, this is too high, so this must be too high, and *this* therefore, too low..." or "If this goes down, then that must go up..." are important insights. Additionally, qualitative measures fit in well with the level and kind of explanation experts make. We did not want to present many arguments which depended on the actual values of measurements in the circuit, because we felt that to do so would be unrealistic in situations where the device was unfamilar to the student. Extensive experience with a particular device may in fact yield student rules like "If the output voltage is between 33 and 35 volts, replace D5." Such rules, while perhaps valid for repair of familiar equipment by average troubleshooters, do not teach concepts and do not generalize to other situations.

We emphasize that our expert is not committed to any single strategy, but to several. For example, if the output voltage of the power supply is high under light load, the expert has the choice of either a conservative, or a more radical, top-level strategy:

A. *Milk-the-front-panel*: Extract as much information as possible from observations of the device under different settings and loads, before proceeding to internal measurements, if necessary. As an important caveat:

A'. *Don't-kill-the-goose*: To keep from blowing the circuit, when milking the front panel, be sure not to set up the panel and put loads on the system that would ordinarily invoke the protective mechanisms of the circuit but that may now be malfunctioning.

B. *Formulate some, though not necessarily all, possible hypotheses:* In particular, don't spend exorbitant cognitive resources on convincing yourself that you have considered all possible hypotheses before checking out the more likely of the ones you have generated.

By offering both approaches, we expose a student to alternative ways of attacking the problem letting him witnessing an expert that is flexible enough to use different strategies for specific reasons.

Finally, the strategies support measurements which, in part, require reasoning about the purposes of each of the function blocks, i.e., teleologically reasoning. This means that each measurement is based on some function-related differentiation of blocks within the device.

A more general top-level strategy is also employed by the expert: split the space of hypotheses. The significant point is that, in order to use this idea, the expert must carry along several competing hypotheses simultaneously. Instead of proposing a single fault and making it the sole consideration of a test, the expert consistently mentions several possibilities among which a given test will differentiate. This kind of strong inference is much more powerful than testing a single hypothesis at a time.

2.1.2 SOPHIE Lab

The SOPHIE lab (part of the original SOPHIE I system) consists of two major components: 1) the automated lab instructor or coach and 2) the simulated laboratory workbench containing various instruments for making measurements and models of both the broken and the working device.

2.1.2.1 The Workbench

The workbench was designed with several pedagogical goals in mind. First it has to encourage students to run their own mini-experiments to explore the workings of the circuit. The mini-experiments consist of first modifying any component in the circuit and then determining the circuit's behavior by performing any set of measurements. To enable this experimentation, the lab provides the capability of saving the current experiment (context) and setting up a new one. Each such experiment forms a layer in a context tree.† Since measurements were easy to make, students could quickly assess the internal workings of the given device with respect to various modifications or faults.

†At any point, the student can push to a new layer and set up a minor variant of his immediate exploration or return back into the middle of a prior experiment. Our goal was to create an environment in which the student could develop a train of thought, with freedom to return and pick up any thread he wished.

An important aspect of the workbench was that it freed the student, and laboratory instructor as well, from having to worry about unforeseen consequences of a proposed modification or fault insertion. Many propagations are sufficiently subtle that even an expert's modifications may unexpectedly blow other components in the device.† This freedom enables students to engage in such useful exercises as finding a fault that does *maximal* damage; an exercise they find intriguing and from which they learn a lot about causality in the circuit.

Understanding causality, at least at the qualitative level of how a faulty component can affect a healthy one, is important. A special mechanism was added to SOPHIE's general-purpose circuit simulator, enabling the overall simulator to capture and portray this qualitative unfolding of causal fault propagation. The mechanism for performing propagation will be discussed further in the knowledge engineering section of the chapter.

2.1.2.2 The Simple Coach

The SOPHIE I system implements a limited coach that performs various checks on the student as he troubleshoots a given fault. The most sophisticated monitoring checks the logical redundancy of his measurements. In particular, the coach has sufficient logical prowess to detect if a proposed measurement *could possibly* contribute any new information about the fault. If not, the measurement is redundant.

The coach is also responsible for checking the logical consistency of all student hypotheses. In the extreme case, if the student offers no fault hypotheses prior to requesting that a component be replaced, SOPHIE does not replace it until he states precisely what he thinks is wrong with it. After the student offers a hypothesis, the coach determines whether it is consistent with the information deducible from his prior measurements. If it is not consistent, the coach identifies which past measurements contradict his current hypothesis. At times we have set up the SOPHIE environment so that if the student, by chance, guesses the actual inserted fault, but his current set of measurements have not yet ruled out other possible ones, the actual fault will be shifted to one of the logically possible remaining ones.

As with any coach, ours tries not to overburden the student with criticism. It uses both a notion of recency and of conspicuousness to choose among the measurements that contradict his hypothesis, leaving it to the student to request older and more subtle ones. In addition, the coach also delineates the measurements that support his hypothesis as well as a third class, those that are consistent with it.

The coach's final activity is to answer student questions. For example, students often want to know if a particular component *could* blow if some other given component were shorted. As we will discuss in section 4, the coach handles its various tasks by, metaphorically speaking, dropping into the background, setting

†One of the initial motivations for building this system stemmed from the need of having an environment into which instructors could insert faults without unexpectedly damaging the rest of the device. This need is enhanced when one considers inserting faults into real equipment such as complex reactors or boilers that can become unstable in surprisingly subtle ways.

up and running its own set of experiments on its copy of the workbench, abstracting the results and formulating a response to the student's query or hypothesis. But to answer the above kind of question, the coach must use its teleological and causal understanding of the device to figure out what boundary conditions (e.g., front panel settings of the device, loads on the device, etc.) would most likely blow the component in question. Thus, in answering this kind of question, the coach exposes the student both to the search for ways of setting up the device and to the consequences of running the device in those situations.

2.1.2.3 SOPHIE Gaming Environment

A game was developed in which one player inserts a fault which another player must then find. The game begins with one person (the "inserter") introducing an arbitrarily chosen fault into the circuit and setting the front panel controls so as to exhibit some external symptom in the device. The other person (the "debugger") must then find the fault by performing a sequence of measurements. Each measurement has an associated cost that roughly reflects the degree of difficulty of making that measurement on a real instrument. For each measurement the debugger makes, the inserter must predict whether the result of the measurement will be higher, lower or approximately the same as that in a working instrument. After the debugger successfully locates the fault, the inserter is given a score of the total cost of the debugger's measurements multiplied by the percentage of times the inserter correctly predicted the outcome. *Thus, it is to the advantage of the inserter to choose a difficult fault but not one so difficult that he can't predict its consequences.* The debugger, of course, attempts to isolate the fault using the least expensive sequence of measurements. The debugger announces a guess by replacing the component believed to be faulted. If the wrong component is replaced, the cost is increased a substantial amount. After each successful fault discovery, the roles are reversed and another game initiated.

The game was designed with two instructional goals in mind. First, we wanted a self-motivating activity that promoted cost-effective troubleshooting. Second, we wanted an activity that required the student to exercise his causal and teleological understanding of the device. To do well at this game requires both kinds of skill. Causal and teleological understanding is forced into application by requiring the inserter to predict the qualitative behavior of the circuit. To begin with, since the inserter must specify how to set the front panel controls to manifest a symptom, he dares not choose a fault whose impacts and implications he cannot project. In addition, he must successfully predict the qualitative response of the circuit at any location measured by his opponent, the debugger. This can deepen his understanding in two major ways. First, if he can guess the troubleshooting strategy that his opponent will choose, he can reflect on his ability to envision the qualititative consequences of a proposed fault at the measurement points favored by his opponent's strategy. Second, when his opponent's measurements diverge from his expectations and are off the direct causal path from the fault to the symptom, he will be forced to think about parts of the circuit he otherwise might not have bothered to master.

An extension of this two-person game to a team game, each team comprised of two people, leads to extraordinarily productive interactions. Teams are more

adventuresome, more willing to probe the limits of their understanding. The discussion of possible ramifications that arise just choosing a fault, for instance, often requires substantial pooling of the team members' information as well as correction of each other's misconceptions. Similarly, when deciding on which next measurement to make, the members will often discuss the possible ramifications of each suggested measurement.

2.1.2.3.1 Using Teams to Generate Protocols on Strategic Thought:

An interesting by-product of using teams is that it provides a beautifully simple "window" into the strategic reasoning patterns and knowledge of the individuals [Brown, Rubinstein and Burton 1976]. The kinds of discussions or arguments that unfold between the team members provide us with detailed information about their strategic reasoning. This includes not only the reasons why a given measurement is being selected but also why some other is *not* being selected. It also helps solve another problem plaguing the general use of protocols in studying reasoning: getting subjects to articulate what they are thinking without becoming self-conscious and thereby altering their natural problem-solving behavior. However, in this situation we find that team players are far less self-conscious than the single informant and that they make and defend hypotheses, plan and revise strategies, and explain things to each other in what appears to be a natural way. Likewise, in collecting a protocol of a subject who is working alone, it is extremely difficult to get insights into why he rejects certain moves; subjects usually feel no need to justify why they don't do something. Instead, the single subject invariably focuses on the more tactical issues of how to accomplish his objective. In the two-person team environment, the arguments that naturally arise involve attempts to justify or defeat a proposed move. The record of these justifications provides a rare opportunity to see strategic reasoning unfold and defended.

2.2 TYPICAL SEQUENCE OF ACTIVITIES WITH SOPHIE

There are many ways in which all the above learning activities might be woven into a coherent training course on electronic troubleshooting. In order to provide a feeling for one such sequence we will briefly describe a twelve hour mini-course that we used to conduct a variety of experiments. The first two hours of this course consisted of a brief introduction to the SOPHIE followed by an intensive study of an instructional booklet reviewing material on basic electronics as related to regulated power supplies, as well as a more detailed explanation of the particular power supply device to be used. This material was rounded out by a question-answering period and an informal discussion on troubleshooting strategies.

In the second period, students alternated between using the expert debugger of SOPHIE II and doing troubleshooting in the SOPHIE lab using preselected faults. Unassisted troubleshooting provided good impetus for paying close attention to what the expert was saying; the students quickly realized that troubleshooting this instrument was harder than it appeared.

The third session consisted of two activities. The first exploited SOPHIE's simulator fully — students were given the task of finding the faults which propagated to blow out a specified component, thus gaining practice in predicting

possible causes of secondary faults. In the second activity, students were joined into two-person teams to engage in more troubleshooting practice.

In the last three-hour period the teams were pitted against each other in the troubleshooting game. After several go-rounds, we broke up the groups and gave each individual some final troubleshooting exercises. For this concluding activity we stressed cost-effective troubleshooting as well as the use of SOPHIE's coach.

2.3 NON-STANDARD USES OF SOPHIE

2.3.1 A Resource Manager

The metaphor of coaching implicitly combines two activities: that of detecting the lack of understanding or faulty reasoning on the part of the student; and that of explaining to the student the critical remedial pieces of knowledge. In numerous cases, detection is considerably easier than remediation. In these situations, it might be advantageous to use SOPHIE in conjunction with a human instructor, with SOPHIE detecting the student's misconceptions and the instructor remediating them.

More generally, consider a laboratory setting with many students and one human laboratory instructor. It is relatively easy to create mock-up workbenches so that any measurement taken by a student while he troubleshoots is monitored and fed into a central SOPHIE coach. When the automated coach detects that a student is stuck in a loop, or is about to make a senseless replacement, then it can notify the instructor to explain the needed remediation to the student. In this way, the instructor need not waste time peering over his students' shoulders; he can count on the system to pinpoint potential opportunities for explanation. He then can employ his own tutoring strategies to decide whether to interrupt the student and, if so, what to say. In this scenario, the automated coach is saving both the instructor's time and effort in that detecting educationally useful moments in a troubleshooting session is very taxing even in a one-on-one situation.

Although the coach in SOPHIE I could perform the important task of detecting instances of inconsistent hypotheses and redundant measurements, a good laboratory instructor should be sensitive not only to mind bugs but also to opportunities to illustrate general principles in particular situations that arise while troubleshooting. As we will see in section 4, SOPHIE III is designed to capitalize on these latter occurrences, and is capable of detecting a wide range of situations that instantiate generic electronic laws. In the same spirit, it can also detect, without a simulation of the piece of equipment, interesting situations directly from measurements made on the broken piece of equipment, itself.

2.3.2 Distributed Instructional Systems and Generative CAI

Considering the cost of the computational resources needed to implement the fully general simulated laboratory and coach, and also wanting to set the stage for exploring versions of the above "resource management" scheme, we implemented a distributed instructional scheme. We used Apple II's, low-cost micro-computers, as work stations coupled to (in principle) a fast mainframe computer containing SOPHIE's general-purpose circuit simulator. The computational capabilities of the mainframe could be used to automatically generate voltage tables characterizing the behavior of the circuit for a *prespecified* set of faults as well as for its normal

working behavior which could then be stored locally on the micro-computers. The micro-computer was inexpensively made to "look" like a simulated workbench by attaching to it a bitpad tablet on which actual color photographs of the desired printed circuit board or schematic were laid. In the micro, a model of the photograph is constructed, making it possible to use the tablet's pen as a simulated voltage probe. With this setup, the student can make any measurement he wishes simply by using the tablet's pen to touch the appropriate terminal on the photograph.

The micro not only simulates the broken (or working) device, but keeps track of all of the student's measurements and attempted component replacements. It can also accurately record the time at which each measurement was made. The student can review the measurements he has taken and ask for prestored help. After he finishes his troubleshooting exercises, the history of his interactions can be transferred to the mainframe and critiqued, off-line, by the SOPHIE coach. The coach can then determine what set of faults the student should try the next day and automatically create the database for the micro's table-driven simulator.

3. Natural Language Engineering

This section provides an overview of the abstractions underlying the natural language interface of the SOPHIE systems and describes lessons we learned using it with students. Readers are referred to [Burton 1976] and [Burton and Brown 1979] for more details.

The goal of SOPHIE's natural language interface is to successfully understand whatever sentences users type into the system. It presupposes that users have a fair amount of knowledge of electronics and are seriously engaged in troubleshooting a circuit. These assumptions are an integral part of the "world view" that has been designed into SOPHIE, limiting the scope of interactions. The natural language interface was built by using an engineering approach that took advantage of these constraints. We developed two techniques: the first was a technique for incorporating the domain semantics into the parsing process that we called "semantic grammars" (because it uses a grammar that is based on semantically meaningful categories). Semantic grammars provide an approach to the construction of efficient, friendly, natural-language-like interfaces. The second technique was a dialogue mechanism that robustly handled the constructs that arise in conversation. We refer to these techniques as natural language *engineering* techniques because their goal is the production of useful widgits, not necessarily the furthering of our understanding of the language-understanding process. In particular, there are many important problems in linguistics and the psychology of language for which semantic grammars are inadequate, and there are classes of dialogue phenomena not addressed by our dialogue mechanism.

The requirements for the natural language interface to an intelligent tutoring system are efficiency and friendliness over the class of sentences that arise in a dialogue situation. The major leverage points that allow us to satisfy these requirements are (a) limited domain and (b) limited activities within that domain. In other words, we know the problem area, the type of problems the students are

trying to solve, and the way they should be thinking about the problem in order to solve it. In SOPHIE's case, the domain is electronic troubleshooting, and the limited activities are taking measurements and developing hypotheses. This is manifested in SOPHIE's view of the world as consisting of an electronics lab with a broken circuit and a virtual lab instructor that can be called upon for criticism or advice.

One representation of the interface to this world is a set of objects and functions or operations upon them. The functional form (a Lisp form) by which one causes things to happen contains a function name (indicating the operation to be performed) and arguments (indicating the objects of the operation). The functional form for measuring the current thru R9, for example, is: (MEASURE CURRENT R9). One view of the task of the natural language interface is to translate the users queries from English into this functional representation. In this view, the functional representation language provides a target output language for the natural language interface. The functional forms which result from parsing the student's statements play a dual role in SOPHIE's operation. In their more obvious role, they are evaluated to carry out the desired operation. In their less obvious role, they serve as data objects that get manipulated by the dialogue mechanism.

As we experimented with early versions of SOPHIE, it quickly became clear that the system needed a model of the dialogue as well as of the lab itself. When the system exhibits intelligence by answering a student's questions or criticizing his ideas, the student invariably assumes the system has intelligent conversational abilities as well. For example, following his question "*what is the base emitter voltage of Q6*?", the student quite naturally asks "*what about Q3*?" This second query, like any containing ellipsed or anaphoric phrases, presupposes a dialogue context. The functional representation for the query contains uninstantiated items which serve as place holders for the meaning that will be gotten from the conversational context. The history list, as will be detailed later, provides the necessary model for the conversational context.

3.1 SEMANTIC GRAMMAR

In semantic grammars, the possible statements to the system are characterized in terms of the underlying concepts (functions and objects) of the domain. The input language is described by a set of grammar rules that give, for each function or object, all possible ways of expressing it in terms of other constituent concepts. Each rule has associated with it a method for building the meaning of its concept from the meanings of the constituent concepts. This allows the semantic interpretation to proceed in parallel with the recognition. Because the method also has the ability to stop the rule if the semantic interpretation fails, the semantics can be used to guide the parser. For example, the rule for <MEASUREMENT> expresses all of the ways in which a student can refer to a measurable quantity and also supply its required arguments. This rule can be used to find out about measurements ("*Is the voltage at the collector of Q5 correct?*") as well as to make them.

3.2 THE DIALOGUE MECHANISM

While involved in a problem-solving session, users assume dialogue context in many of their statements. The following statements occur naturally and require context to understand:

{In a context where the student has mentioned Q6}
What is the voltage at the base?
At the collector?
What about the current through the emitter?
Is that right?
What about Q5?

SOPHIE's dialogue mechanism accepts these. Viewed from the functional representation standpoint, the mechanism basically permits argument replacement, function-name re-use and argument re-use. These operations correspond to powerful, frequently occurring dialogue constructs including forms of pronominal reference, anaphorae and ellipsis. *"What is the voltage at the base?"* and *"set it to 50 ohms"* are examples of functions missing arguments. *"What about Q5?"* and *"thru R9?"* are examples of arguments missing surrounding forms. *"At the collector"* is an example of both a missing argument (to the function COLLECTOR) and a missing surrounding form (a function to apply to the collector of a transistor).

The functional representations returned by the parser for these sentences contain place holders for the missing pieces of information. For example, the functional form representation for *"What about Q3?"* is (REFERENCE Q3 ?). REFERENCE occurs when there is an argument in need of a surrounding form. The "?" position provides a place holder for the "meaning" of this ellipsis and is filled in with the functional form of that meaning when it is determined from the dialogue context. The functional representation for *"What is the voltage at the base?"* is (MEASURE VOLTAGE (BASE (PROREFERENCE (TRANSISTOR) ?))). In this form, PROREFERENCE marks that something has been deleted or pronominalized and "?" provides a place holder for the deleted construct. The second argument contains type information about the deleted construct that is determined from the sentence context. (In this case the list of acceptable types has only one element, TRANSISTOR, because transistors are the only parts which have BASEs.)

The candidates for the missing parts of a statement are supplied by the dialogue context model, a list of the previous functional forms. The search for the missing items is guided by the type information either taken from the given object or supplied in the functional form. Although the view of dialogue constructs as supplying functional forms and arguments handles a large, useful class of queries robustly, there are many classes of dialogue phenomena it does not seek to deal. See [Webber 1979] and [Grosz 1978] for more general approaches.

3.3 IMPORTANCE OF PARAPHRASE; A SIMPLE EXPERIMENT

Having a system recognize alternative wordings of the same thought is important because our experience indicates that individuals; 1) phrase questions in a variety of ways and 2) they are poor at paraphrasing. In an informal experiment to test the habitability of a system, we asked a group of students to write down as many forms as possible of asking a particular question. The students each came up with

one phrasing very quickly but had difficulty thinking of any others. Surprisingly, nearly all of their first phrasings were different, indicating the difficulty of accepting the first phrasing in all cases.

3.4 EXPERIENCES

The natural language interface to SOPHIE took approximately two man-years of effort, and evolved over the course of four years. Although extensions were made as the capabilites of SOPHIE increased, the primary impetus for change was the difficulties encountered by users. The dialogue mechanism is one feature that was added as a result of users' experiences. All during the use of the natural language interface, any query not understood was automatically sent as a message to one of the designers. This facility allowed the monitoring of SOPHIE's use, even at remote sites on the ARPA network. The non-comprehended sentences were saved and used as a "test" set for new modifications to the grammar. In the end, the natural language front end reached the state of being easily usable by electronics students. The entire system was fast, taking less than 150 milliseconds for almost all inputs, and robust, correctly handling more than 90 percent of queries.

Abbreviations and spelling correction are important to reduce the required amount of typing. This is not to say that users would be happier with an abbreviated command language; naive users tend to use full English queries. But as the naive user becomes involved in the troubleshooting process, he occasionally tries a new (often shorter) phrasing, an ellipsis or an abbreviation. If the system accepts it, the user incorporates the phrasing into the known accepted language base and continues to use it. (Although it is interesting to note that when, due to heavy system load, response time slowed down to more than about 5 seconds, the dialogue constructs disappeared and the student's format reverted to that of complete sentences.) The critical concept is that the system allow users to move from complete sentences to their own particular short command languages without rejecting very many intermediate statements along the way. The continuum from full sentences to cryptic, ellipsed abbreviations allows users to find their own comfortable method of expression.

When a user's query is not completely specified, or ambiguously misspelled, the system should provide unambiguous indication of what it has changed or overlooked to understand his request. The user may have meant something else and be very much misled by the response. For example in the following excerpt:

What is the current through R11?

Across R9?

the system interprets the second query as asking for the current through R9. The student, however, may be asking for the voltage across R9 rather than the current thru it; believing that the semantics of "across" would override the recent mention of the quanitity "current." To forestall misunderstanding, SOPHIE always responds to every query with a complete sentence describing the query as well as the answer. In this case, SOPHIE would respond with *"The current through R9 is 10 milliamps."* This solution does not work all the time; apparently the users' predispositions toward their own hypotheses sometimes override their ability to read/perceive what is being said, leading them to overlook all except the part of the response that is consistent with what they believed their query to be.

The system should have and present to the user a consistent model of its capabilities. Serious problems arise when the user does not have an acccurate model of the limits of the system. If the user thinks the system has more capability than it does, he will continually be asking things that the system can't understand. If the user does not know about a system capability, he will never use it. A natural language interface does not help because the interface does not provide any scope to the system. With a menu interface, the user has at least some idea of the capabilities of the system. In SOPHIE we attacked this problem by having the natural language component recognize common misconceptions about its world view well enough to provide remediation. An example is given in the following student query and system response:

>> COULD Q1 OR Q2 BE SHORTED?

I can only handle one question, hypothesis, etc. at a time. The fact that you say "OR" indicates that you may be trying to express two concepts in the same sentence. Maybe you can break your statement into two or more simple ones.

Another more complex example is:

>> WHAT IS THE CURRENT THROUGH NODE 4?

The current through a node is not meaningful since by Kirchoff's law the sum of currents through any node is zero. Currents can be measured through parts (e.g., CURRENT THROUGH C6) or terminals (e.g., CURRENT THROUGH THE COLLECTOR OF Q2).

The system response to the question addresses the student's misconception of where current is measured and, more importantly, gives the user some indication of how to ask for the information he wants; because C6 and the collector of Q2 are two of the terminals connected to node 4.

There are several limitations about the domain that made it appropriate to the semantic grammar technique. Since the user interacts with the system in the same way he would interact with a laboratory, few questions involve quantifiers. Also, relative clauses are not an appropriate way of asking direct measurement questions. These limitations are not inherent in the semantic grammar framework. More complex constructs, in particular quantification, have been implemented in applications of LIFER (Hendrix 1977) which uses a similar framework. In addition, the framework has been applied to other domains, such as in ACE (Sleeman and Hendley 1981) which analyzes student explanations about NMR spectra interpretation. RUS [Bobrow 1978] which began as an attempt to look into domain-independent semantic grammars, has developed a redefinition of the role of syntax in natural language and resulted in an efficient method for mixing syntax and semantics.

3.5 GRAMMAR REPRESENTATIONS

Two representations for semantic grammars were explored in SOPHIE. The first was the direct encoding of the grammar in Lisp code. In this encoding, each grammar rule was hand-coded as a Lisp function that typically checked for certain words or classes of words in the input, advanced the input, called other Lisp functions to look for other non-terminals, and built the interpretation of the rule. This has the advantage of not requiring a separate grammar formalism or parser, thus both reducing the overhead of getting started and allowing use of the powerful INTERLISP debugging environment in developing the grammar.

In this formalism, *fuzzy parsing*, the ability to skip words in the input sentence, was explored. When few and unsophisticated concepts were accepted, almost anything with the right words could be taken to mean the same thing, allowing recognition of some sentences that would not have been otherwise recognizable. As the complexity of the phrasings increased though, particularly with the inclusion of ellipsis recognition, the fuzziness led to troublesome misunderstandings. With two important exceptions, our experience showed that fuzziness caused more problems than it solved. The first exception is the fuzziness technique of ignoring words that are not in the system's dictionary. By informing the user of unknown words and trying to make sense of the remainder, the system can bypass many adjectives, adverbs and expletives. The second exception is the technique of ignoring words that are left over after a successful parse. The user should, of course, be told which words are being ignored.

The second representation for semantic grammars arose out of necessity. As the grammar became more complex, the Lisp implementation became hard to understand, so the grammar was translated into the *augmented transistion network* (ATN) formalism. The ATN has the advantages of being more concise, having factored expressive power and having a better development environment. In our ATNs, the structure built on the arcs and returns from the final state, is the "semantic interpretation" of the sentence, not its syntactic structure.

An initial efficiency advantage which the direct Lisp encoding had over the ATN, being executable directly as machine code, was overcome by the construction of an ATN compiler [Burton 1976, and Burton and Woods 1976]. The ATN formalism can be used to develop a range of interfaces, from simple keyword interfaces to highly complex, general schemes, with very little loss of efficiency. In addition, since the code produced by the compiler does not require the large Lisp environment, the compiler can be straightforwardly changed to produce code for microcomputers. This would allow the ATN formalism to be used for interfaces that are developed in a large machine environment but run as part of systems on small machines.

4. Knowledge-Engineering Techniques

Our intent in this section is to focus on the techniques that we have used to construct efficient and robust intelligent tutoring systems.† Achieving both efficiency and robustness is of paramount importance. Efficiency is needed because the attention span of students, even highly motivated ones, is often considerably shorter than that of the typical user of other knowledge-based expert systems; and robustness is needed because the questions asked, hypotheses proposed and experiments tried by students just learning a subject often are (or appear) bizarre.

We will discuss here two systems, SOPHIE I and SOPHIE III. Since most of the knowledge-engineering techniques in SOPHIE I were published some time ago

†By robustness, we mean able to work under many circumstances; particularly, providing an appropriate response to an unanticipated student action.

[Brown and Burton 1975, Brown, Burton and Bell 1974], we will limit the discussion of SOPHIE I to (1) the central abstractions of that work and (2) some of the misconceptions that surround the use of simulations as an inference tool. Understanding the strengths and inherent weaknesses of SOPHIE I will also set the stage for describing SOPHIE III. The later system reflects a major departure from SOPHIE I in its underlying knowledge-engineering techniques.

The "expert troubleshooter" in SOPHIE II focused on the problem of good explanations of problem-solving knowledge. Its "intelligence" resulted from straightforward applications of well-known knowledge-engineering techniques, primarily decision trees, and thus will not be discussed further.

4.1 SIMULATION-BASED INFERENCE SCHEMES: SOPHIE I

SOPHIE I utilized a simple but powerful idea that had already proved effective in one of the first major expert systems, Dendral [Lindsay, et al., 1980]. Both SOPHIE I and Dendral achieve their logical prowess by obtaining leverage from some powerful and robust non-AI tools, augmented by collections of rules (or procedural specialists) that "intelligently" use the tools and their results. In Dendral's case, the primary tool is an elegant mathematical algorithm for enumerating all the subgraphs of a graph without generating isomorphs. In our case the primary tool is the general-purpose circuit simulator, SPICE, which is in continued general use by circuit designers. Various knowledge-based specialists in SOPHIE I transform a given problem into a set of subproblems, each of which can be solved by executing an appropriate model on the simulator, critiquing the intermediate results, and infering an answer to the original problem from the resulting data. In other words, SOPHIE I, when presented with a question to answer, a hypothesis to evaluate, or a measurement to check for redundancy, sets up its own set of "experiments" and from the results, infers an answer to the question. There are, however, numerous problems merely in determining the precise experiment(s) necessary to gather sufficient information for infering an answer.

The following few sections will explore both the strengths and weaknesses of using simulations as the core of an inference engine. We will also discuss some unusual ways that simulation can be used in inference tasks.

4.1.1 Strength and Weaknesses of Simulations

There are two primary reasons for exploring ways to use simulation as part of an inference scheme in the domain of electronics: 1) there is a vast amount of electronic expertise already neatly encoded in simulation models, particularly for the generic components (diode, transistor, etc.); and 2) behind the simulation algorithms, there is a vast amount of mathematical expertise that efficiently implements a calculus for "reasoning" with these models. We are given a strong domain model; we need only find clever ways of using it.

Robustness Requirements When Using Simulations for Inference

4.1.1.1 Implicit Assumptions in the Models

The most characteristic shortcomings of nearly all simulations is that they have a large number of presuppositions built into them, many of which are implicit even

to their designers. Applying a simulation in a new context can be problematic since its implicit assumptions may be violated. This is especially true when using simulations to reason about potential consequences of faults (hypothetical or real), because faults can so drastically alter the internal workings of a system that the faulted system violates even the most basic, common-sense rules of any intentionally designed artifact (e.g. man-made device). In such cases, not only are the implicit assumptions of the original simulated system apt to be violated but even the mathematical techniques being used to "solve" or run the simulation can have their assumptions violated.

These observations are not limited to just electronic simulations. In fact, they are more likely to occur in other domains (reactors, hydraulic systems, and weather models, for example) where the simulation models are more likely to be tied to an already existing physical system. In such cases, the designers of these simulations are apt to let their understanding of the overall system suggest what aspects of the system's individual components need be captured. This leads to ad hoc models which subtlely depend on the system working as intended [see deKleer and Brown 1981 for an extensive discussion of this point, especially as it applies to robust mental models].

It is difficult to predict whether a proposed context on which to execute the simulation or a modification to its structure will transgress the assumptions of the model. Worse, it is not always possible to detect, even after running the simulation, that a transgression has occurred; the model may still produce plausible behavior which is, in fact, wrong. Only if a trangression manifests itself by having the simulation "diverge" (so that no solution is found) is the state easy to detect.

The need to make explicit all the caveats underlying the overall simulation model — or, at least, to characterize its domain of applicability — is more crucial for the use of simulations as an AI tool than for their normal use. In normal use, it is often the designer who invokes the models on a new, perhaps hypothetical, situation and who will often intuitively know if an implicit assumption is being violated. But an AI system, such as SOPHIE I, generates deep within its inference chain hypothetical modifictions to a model that may make little sense to the expert and hence violate all kinds of implicit assumptions. In addition, an obvious use of an "intelligent" shell around a simulator is to have it rapidly generate a vast collection of hypothetical situations which is apt to stress the boundaries of the models in novel ways. In both of these cases, reliable use of the simulation models mandates that the assumptions/caveats be explict and checkable.

Fortunately, the general-purpose simulator that was used in SOPHIE I had relatively few bothersome, implicit assumptions (although our own extensions to it and the various circuit specific, functional simulations built especially for SOPHIE were plagued with such problems). One proposed counter-assumptional experiment had an external voltage source (a fully functional storage battery) being hooked up to the *output* terminals of the regulated power supply. Such a modification had never occurred to us because it violates the core meaning or use

of a power supply: to deliver voltage, not absorb it.† It turned out that we had built into the functional simulator a teleological assumption that the system was in either a current-limiting or a voltage-limiting mode — a reasonable asumption for a regulated power supply. But, with the battery hooked to its output terminals, it was no longer functioning even as a power supply, let alone a regulated one. Although the general-purpose simulator could handle this experiment, the functional simulator, hand-tailored to allow extremely fast simulations of the given power supply, completely failed to model, even qualitatively, what actually happens.

4.1.1.2 Models of Fault Modes

Another complicating factor in using simulations, especially to handle hypothetically faulty components, is the difficulty of characterizing all the possible behaviors of a malfunctioning component. This difficulty becomes accentuated in dealing with the models for primitive components since the program or mathematical transfer function for a primitive is often structurally opaque. The opacity results from the fact that the transfer function is derived empirically and hence its internal structure doesn't necessarily stand in any meaningful relationship to the internal structure of the component; perturbations to the internal structure of the component (faults) in this case have no obvious correspondence to perturbations of its model. This makes using the working model of the component as a guide for constructing the faulty model a problematic venture.

4.1.2 Simulation Used as Part of the Inference Machinery

For many of the capabilities needed by SOPHIE I, simulation is only part of the solution. In the next section we shall describe simulation capabilities, how simulations fit into the solution and what auxilliary mechanisms were added.

4.1.2.1 The Need to Capture Causality

The kinds of simulators most often used in electronics rely on "relaxation" methods for solving the system of equations characterizing circuit behavior. Although these techniques are extraordinarily powerful, the intermediate solution states they pass through bear little resemblance to any kind of causality underlying the circuit. Indeed, the circuit itself is represented as a set of constraints and the sought-after behavior is that which *simultaneously* satisfies all the given constraints. Nevertheless, students and experts alike can best understand a circuit, and remember their understanding, if an "explanation" reveals some of the underlying causality. Thus, to be able to get maximal pedagogical leverage from the simulation, we need to find ways of getting causal information. Two possible strategies come to mind: (1) deduce some of the circuit's underlying causality from the solution space of the constraint equations; or (2) factor the system of constraints and simulation methods so that the relevant parts of the system's

†Of course from another perspective this experiment is not quite so absurd since one might use a regulated power supply to charge a battery and one might also make the mistake of trying to charge a completely charged battery.

underlying causality are not hidden by relaxation. In SOPHIE I, we chose the second alternative.

4.1.2.1.1 Capturing Causality

As a first step, the appropriate level of causal abstraction must be identified. In SOPHIE's case the most important aspect of causality pertains to fault propagations: if a component faults, what other components will blow as a result? We factor the simulator's constraints into two classes, one comprising the normal constraints of electronic laws, the circuit's topology and its component values, and the other comprising *meta-constraints* characterizing the conditions at which components will blow, e.g. power dissipation factors etc.†

This factoring into constraints and meta-constraints allows the relaxation-based simulator to be augmented with two local procedural experts or specialists. The first specialist isolates all violations of the meta-constraints in the database produced by the simulator and if it locates more than one component that is overstressed it determines which one of these will blow first. Then, a second domain specialist determines how that component will fault; whether it will open, short or beta shift. The simulation model of the circuit is then modified to reflect this additional fault and the simulator is again called to construct a new layer in the database. The whole process is recursively repeated until the simulator produces a database that satisfies all of the meta-contraints. This final database is then returned along with the sequence of fault propagations that led to it. This combined structure contains the desired causality of fault propagation and the fixed point of the constraint equations, each being generated by a technique optimized for the task. This composite structure can then be used to answer questions about the consequences of the proposed hypothetical modification, to generate causal explanations concerning fault propagation or, perhaps, to drive a graphics system to animate the propagation.

4.1.2.2 Determining "Interesting" Experiments to Run

Answering certain classes of questions requires not only the ability to *run* the simulation model but also the ability to reason *about* the model itself. The need for both is most easily seen by considering questions that contain an implicit existential quantifier. Take, for example, the seemingly simple question "If X is shorted will Y blow?" The implied quantifier becomes clear in the following rephrasing: "Does there exist a boundary condition for the simulation such that, if X is shorted, it will lead to Y being blown?" The problem here is analogous to the classical problem of program testing: choosing a set of test data such that if the program has a bug, it will be manifested on that data. This analogy also illustrates

†This provides another subtle example of an implicit assumption built into the simulation. The mere fact that we choose excess power dissipation as being the significant feature underlying a component's failure through fault propagation signifies a belief that in the circuits we are handling, very fast but short lived potential spikes won't exist. In normal power supplies, that is a fair assumption. But, if such spikes are present, transistors will fail when the potential of the spike is great enough, even though its power is arbitrarily small. If a component were to fail so that it generated such a pulse or pulse stream, this assumption would be violated, leading to a potential divergence between the model and the physical system.

that just running more successful experiments or data sets does not necessarily improve the chances that the program is bug-free. Running even an infinite number of test cases, all of which exercise the same aspects of the program, will not guarantee more than does just a single experiment. The test cases must be independent with respect to the operation of the program. This means that what makes one experiment or data set "logically" different from another depends on the logic of the program.

Although the above observation is well known in programming methodology, its analogy in the use of simulations is not so well appreciated. Much of the power in using simulations as an inference engine stems from heuristics for determining "interesting" boundary conditions for the simulation.

Given the question "If X shorts will Y blow?", the heuristic calculus used to construct a set of boundary conditions first determines the function blocks containing the components X and Y. The calculus then determines how to set up the loads and control settings on the device so as to activate those functional blocks.

4.1.2.3 Hypothesis Evaluation

In SOPHIE I, evaluation of a hypothesis is the process of determining its logical consistency with respect to the information derivable from the current set of measurements.† Note that a hypothesis can be logically consistent with the known information and still not be what is actually wrong with the circuit. For example, if no measurements have been performed then any syntactically correct hypothesis is acceptable.

For a given hypothesis, the hypothesis evaluation specialist partitions the current set of measurements into three classes: one containing the measurements contradicted by the given hypothesis, another containing the measurements logically entailed by the hypothesis and the last class containing those measurements independent of the logical consequences of the given hypothesis.

These partitions are determined by taking into consideration all the logical implications derivable from the given hypothesis. If, for example, the hypothesis concerns Q3's base being open, there need be no direct or obvious measurement on Q3 that indicates whether it is working or not. By taking into consideration both the local and global interactions of components in the circuit, measurements arbitrarily far away from Q3 may be used to support or refute the hypothesis.

Simulation can be used to determine the consequences of a hypothesis much as it was used to infer the consequences of the assertional part of a hypothetical question. In particular, it is used to construct a "hypothetical world" specified by the hypothesis (e.g., "Q3 must have its base open"), and in that hypothetical world, all the student's measurements are "replayed." If the values of any of these measurements are not *qualitatively* equivalent to the ones observed by the student in the faulted circuit, then a counterexample or inconsistency has been established.

†A hypothesis concerns the state of a given component: a capacitor being shorted, a resistor being open, a transistor being shorted, etc.

4.1.2.4 Hypothesis Generation (Theory Formation)

The method of constructing the set of those hypotheses or faults consistent with the currently observed behavior of the faulted device uses the venerable "generate and test" paradigm. First, a backward-working specialist, the PROPOSER, examines the value observed for an external measurement and, from that observation, determines a list of all possible significant hypotheses which explain it. Because the PROPOSER is not endowed with enough knowledge to capture all the complex interactions and subtleties of the circuit, it can err by including hypotheses inconsistent with the observed behavior.

A second specialist, the REFINER, "simulates" each fault on the current hypothesis list to check whether it explains all the measurements that the student has taken. The REFINER must take into consideration all of the complex interactions overlooked by a simple theory of the circuit. Depending on the REFINER's simulator to check out all the subtle consequences of a proposed hypothesis, however, leads to a major problem; in order for a hypothesis to be simulated, it must be a fully instantiated fault. The hypothesis "the beta of the Darlington amplifer is low," for example, does not make clear *what* the proposed beta is, just that it is lower than it should be.

It is the job of a third specialist, called the INSTANTIATOR, to take an underspecified fault and instantiate it. The INSTANTIATOR uses several techniques to determine a potentially consistent specification of an underspecified fault. The most general of these techniques is a simple hill-climbing strategy in which a specific value for the fault "schema" is guessed and then partially simulated (enough to determine the external behavior). From the result of that simulation another guess is made, until finally a value is found that causes the desired behavior. Because this strategy executes the simulation many times, a special-purpose, functional simulation model is used that runs several orders of magnitude faster than the general-purpose simulator.

4.1.2.5 Determining Redundant Measurements

Perhaps the most complex logical task performed by SOPHIE I is the task of verifying whether or not a given measurement *could* possibly add any new information about what might be at fault.

Our approach to handling this task stems from an analogy for how one might prove the independence of an axiom set. If one is trying to establish that a new axiom is independent from a given collection of axioms, one might try to construct a "possible world" for the original axiom set which is not a possible world for the augmented axiom set.

In our case, we view each measurement-value pair as an axiom or assertion and proceed according to the above analogy. The set of hypotheses constructed by the hypothesis generator serves as the set of all possible world models which are consistent with the known measurements.

Although this technique has proved to be very powerful, for many pedagogical uses it is too powerful.† Some of the redundant measurements it detects can be

†This is the one area in which SOPHIE repeatedly surpasses its designers.

explained by a single step argument. However, others it detects equally easily take a large number of "logical" steps to explain. The disadvantage of the technique is that it can't distinguish between these two cases. What is needed is a human-like reasoning scheme capable of detecting redundancies on its own. This leads to our discussion of SOPHIE III.

4.2 SOPHIE III

The basic theme of the overall SOPHIE project has been to avoid restricting the actions of the student while allowing the computer to act as a coach, to assess the student's understanding and to guide him to a better understanding of electronics and troubleshooting. Building such laboratory requires a system which (1) allows the student to take the initiative, (2) is able to make significant inferences about the circuit from the student's measurements, and (3) is able to explain those inferences.

Meeting these requirements individually is difficult; in SOPHIE III we wanted to achieve them simultaneously. SOPHIE I allowed the student complete freedom of measurement and employed an extremely powerful inference strategy, but it could not explain the reasoning behind many of its inferences. SOPHIE II, on the other hand allowed the student very little initiative, used only a simple inference strategy, but was capable of very subtle explanations of its electronics and troubleshooting deductions.

	STUDENT INITIATIVE	INFERENCES	EXPLANATION
I	Yes	Powerful	Poor
II	Very Little	Weak	Good
III	Yes	Powerful	Good

Our approach in constructing SOPHIE III was to base its inference techniques on those that we observed experts and students using. By making SOPHIE's inferencing strategies more akin to those used by the student, we could begin to determine which deductions the student was using, construct a model of his abilities, then use this model to generate explanations in familiar terms.

We also needed a fundamentally different and more human-oriented inference scheme because we wanted to investigate using SOPHIE as a computer-based consultant for on-job-site training as an intelligent job-performance aid. We thus wanted SOPHIE III to be able to work from measurements being performed on *real, physical equipment*, completely independently of any kind of circuit simulator.

4.2.1 Knowledge Engineering Perspective

The knowledge-engineering task of building SOPHIE III has several interesting features. First, the pedagogical emphasis demands an unusual amount of attention be paid to explanation of deductions. Because one goal is to help the student learn, explanation must be considered equally as important as deduction. Consequently, more of SOPHIE III's computational resources are spent in recording deductions and

manipulating justifications with their underlying *assumptions* than in actually making the deductions in the first place.

Another capability not usually required by knowledge-engineering systems is that SOPHIE III must be able to hypothesize and reason about a student's partial understanding of electronics and troubleshooting. To do so, it must possess many logically redundant strategies since it can't assume the student will know the best ones. Thus many conclusions it produces have numerous independent proofs, each with its own set of justifications and assumptions.

A third complication facing SOPHIE III stems from the fact that electronics (and electronic troubleshooting) is a complex problem domain, part of which has been formalized, part of which has not, especially in terms of the causal calculii tacitly used by human experts. Because of the complexity, we were faced with either restricting ourselves to formalized aspects of the domain or working out a framework to systematically include the collection of ad hoc rules and inference mechanisms needed for the poorly understood part of the domain. We chose the latter course, and our strategy for building the electronics expert was to encode as much of the knowledge in the most general form possible. To complement the general knowledge, circuit-specific knowledge provided by electrical engineering needs to be included in the expert. The explanations of the circuit-specific knowledge, are supplied by the electrical engineer and are attached to each knowledge rule. In contrast, explanations for deductions made from the generic electronic laws do not presume any particular circuit and thus can be constructed from the steps taken by the inference mechanism used to make the deduction.

It would have been easier to encode much of the general knowledge as circuit-specific knowledge. We chose not to for three reasons. First, general knowledge has a very parsimonious structure making it relatively easy to analyze its limitations. Second, the more powerful the general knowledge, the less circuit-specific knowledge is necessary. Third, circuit-specific knowledge, since it is encoded from the engineer's particular analysis, is liable to undermine robustness since it reflects his underlying assumptions.

In SOPHIE III the circuit-specific knowledge is only employed after the general knowledge has made as many deductions as possible. Thus, the circuit-specific knowledge is aimed directly at those situations in which the general knowledge fails. Because of its known limited context, circuit-specific knowledge can be put in a canonical form, the number of ad hoc rules can be minimized and the determination of whether enough circuit-specific knowledge has been included to succeed is made much easier.

The generic electronic knowledge is solely concerned with the states and the behaviors (voltages and currents) of primitive components (e.g., diodes, transistors). This general knowledge is used by an inference scheme based on propagation of constraints which deals with simple inequalities rather than of single numeric quantities. Although, this feature complicates the constraint propagation, it both makes it possible to handle many kinds of inferences humans actualy perform and provides a rudimentary notation for expressing *qualitative* constructs.

The circuit-specific knowledge is organized around a structural decomposition of the circuit. A circuit is a designed artifact, consisting of a collection of weakly

interacting modules. Each module is, itself, a circuit in its own right and can be likewise decomposed. The modules behave cooperatively to produce the behavior of the overall circuit. The key to the circuit-specific knowledge is having terms to express the behavior of these modules. The circuit specific knowledge consists of rules about how the behaviors of modules affect the behaviors of other modules: Neighboring modules affect each other as well a being influenced by the behavior of their lower-level, constituent modules. Constructing the right kind of knowledge structure which, in part, reflects a structural decompostion of the given circuit enables these rules to be systematically encoded where one kind of rule forms the inheritance in the knowlege structure and the other kind of rule forms transformation between "neighbors" of the structure.

Perhaps surprisingly, neither the general nor circuit specific reasoning engine has any direct knowledge of troubleshooting. The troubleshooter is a separate and general system, whose knowledge base is extremely small and is totally dependent on the electronics expert module. The troubleshooting module examines the deductions which the electronics expert does or could potentially make. This requires that it has intimate familiarity with both the electronics expert databases and inference mechanisms. The troubleshooter's simplicity results from the fact that the electronics expert module explicitly records each assumption it makes, and these are scrutinized by the troubleshooter in order to determine why the circuit is not functioning as it should.

The existence of the parasitic troubleshooter does not affect the deductions the electronics expert can make, but it does require that every deduction be augmented by a precise justification recording the inference mechanism and circuit data used. The success of the troubleshooter stems from the fact that in the troubleshooting task some component, necessarily, is not behaving as it should. If a component is not behaving as it should the electronics expert will eventually encounter a irreconcilable contradiction since its model of the circuit will differ from the actual faulty instance that is being debugged. Every inference the electronics expert makes, implicitly involves some assumption (the component is behaving as specified by its manufacturer) about a component or piece of circuit wiring. When the circuit contain a fault, these assumptions must be made explicit. In troubleshooting, a contradiction is an extremely informative event since it indicates which assumptions are violated, narrowing the field of possible faulty components. The entire troubleshooter is based on this one idea, an idea that will be substantally expanded later in this chapter.

The cost incurred by this elegant scheme is that the inference mechanism of the electronics expert must always supply accurate and exhaustive justifications and assumptions for all of its deductions. A single exception could render the troubleshooter impotent. To some extent this same organizational cleanliness is also demanded for generating coherent explanations, but the added constraint of this troubleshooting scheme now necessitates that the electronics expert's deductions be void of any hidden presuppositions.

4.2.2 The Architecture of SOPHIE III

SOPHIE III consists of three major expert modules: the electronic expert (which

consists of two submodules), the troubleshooter and the coach. The electronics expert which received, by far, the most work utilizes mostly general electronic knowledge with some circuit-specific knowledge (in this case, regarding the IP-28 regulated power supply). One of its submodules, LOCAL utilizes the general knowledge and contains, among other things, a local constraint propagator, hence its name. The other submodule, CIRCUIT, is responsible for handling all the circuit-specific knowledge. The other two experts, the coach and the troubleshooter, are both self-contained and circuit-independent. The coaching expert examines those deductions to determine whether or not to interrupt or advise the student. The coaching expert is only partially implemented; the biggest unimplemented portion is the user model and interruption capabilities. The troubleshooting expert is relatively small, using the electronics expert to evaluate hypothetical measurements, and then choosing the most informative one. All three experts operate using only the information known to the student (measurements and schematic), from which they make as many (often redundant) deductions as possible.

The architecture of the electronics expert is a composite of several very different inference techniques. The circuit-specific reasoning can be roughly broken into two types. The first is a rule-based system which uses voltages and currents, propagated by LOCAL from the current set of measurements to determine the behaviors of circuit modules. The second mechanism determines what the component fault modes can cause the symptomatic module behaviors. The fundamental problem of intercommunication between different reasoning types is elegantly solved in SOPHIE III with a common language of justifications and assumptions: Each deduction made by any of the reasoning types simply records the reasons for and assumptions under which the deduction was made. This justification/assumption database, just as in a general-purpose truth-maintanence system, can be oblivious to the different kinds of reasoning that underlie each deduction step. This is also the database from which the troubleshooting expert works.

Three types of reasoning are involved in the electronics expert each with its own knowledge structure, complete with inference mechanism and database (see Fig. 4.1). The propagation database contains quantitative deductions about voltage and currents (e.g., output voltage is 30 volts) made by using the models of the components. The qualitative database contains assertions about the operating regions of the components, voltages, and currents (e.g., output current is low, transistor Q5 is off). The database for the third knowledge structure, the behavior-tree, consists of the possible behavioral modes of components and circuit modules (e.g., R5 is open, the current source is anemic).

4.2.3 The Local Propagator

The local propagator (LOCAL)† forms the basis for the electronics expert. It uses general knowledge of circuit laws to determine what further voltage and current consequences can be determined from the measurements that have been made. The deductions of the local propagator are used not only for their simple explanatory

†The propagator used in SOPHIE III is similar to Stallman and Sussman's EL which is also based on propagation of constraints [Stallman & Sussman 77]. LOCAL differs from EL in that it uses assumptions more widely and is capable of manipulating simple forms of inequalities.

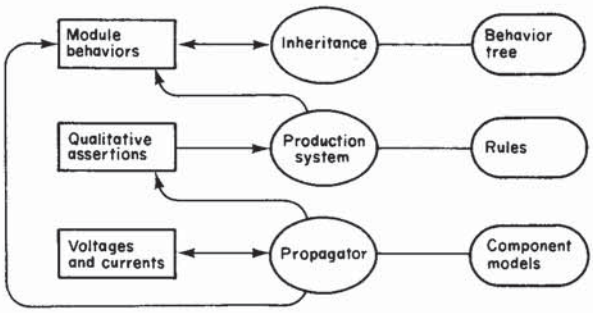

FIG. 4.1. Electronics Expert

capability, but for reference by the other two experts. In this section we will focus on the troubleshooting consequences of a measurement. In order for a measurement to provide useful information we must have some expectation about its value.

This local propagator may appear simple, but the variety of subtle problems with which it must grapple make it quite complex. The biggest obstacle is the necessity for generality — it must deal with arbitrary measurements in arbitrary circuits. But the profit we gain is great; because it is the only part of SOPHIE III that has to reason upon the measurements and circuit topology directly, it greatly simplifies the other experts which need only refer to LOCAL's deductions.

The propagations of even a single measurement can be quite deep. Suppose the resistor we have been using as an example occurred in the circuit Fig. 4.2. Suppose we measured the current in R13 to be 1 milliampere. Since the resistance of R13 is 100 ohms, the voltage across it must be .1 volts. But the voltage across R13 is the same as the voltage across the base-emitter junction of Q6. We know that a silicon transistor conducts no current when the base-emitter voltage is less than .6 volts so the current flowing in each of the resistor terminals must be zero. Now consider the node N9 which connects to R13, Q6 and R16. The current flowing out of R13 and Q6 must be flowing into resistor R16 so the current flowing through R16 is 1 ma. As just shown, a single measurement can lead to the calculation of a lot of information about other circuit values. The local propagator of SOPHIE III is an attempt to formalize this "If $A = x$, then $B = y$" kind of reasoning. It keeps records of each deduction so that explanations can be generated and examined when troubleshooting.

Although the concept of propagation applies to both AC and DC circuits, at present SOPHIE III only models DC behavior. This is sufficient to model the quiescent behavior of circuits, and adequate for modelling power supplies since they (except

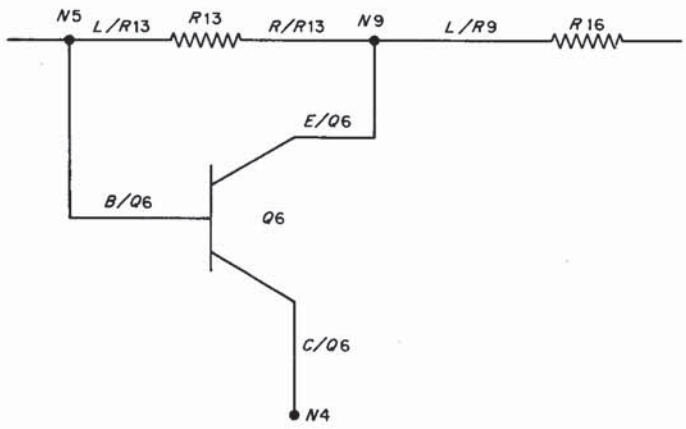

FIG. 4.2. A Current Limiter

for switching regulators) can be understood almost entirely from the quiescent point of view. The circuits consist of resistors, diodes, zener diodes, capacitors, transistors, switches, potentiometers and DC voltage sources. Each component has a expert associated with it which constantly checks to see whether the currents or voltages on its terminals are measured or calculated; as soon as a value is discovered for one, it is used to deduce voltages and currents on the others. Since the terminals and nodes are shared with other devices, these new currents and voltages in turn trigger other component experts. The process continues until no new information can be determined.

In order to explain propagations, a record is kept of which device's expert made the deduction. Propagations are represented as:

```
(<type> <location> (<reason> <component> <arg>) <assumptions>).
```

The <type> is either "VOLTAGE" or "CURRENT." The <location> is a pair of nodes for a voltage, a terminal for a current. The <reason> is a token describing how the device expert made the propagation.

The circuit consists of components whose terminals are joined (two or more) at nodes. Since terminals, unlike nodes, are always attached to components we adopt the convention of labelling them by <terminal-type>/<component>. Currents are normally associated with terminals, voltages with nodes. In Fig. 4.2, L/R13, R/R13, B/Q6, E/Q6, and C/Q6 are terminals, and N5 and N9 are nodes.

The simplest propagations are those involving the Kirchoff voltage and current laws. Kirchoff's current law states that if all but one of the terminal currents of a component or node are known, the remaining one can be deduced:

```
(CURRENT T/1 (MEASUREMENT)) = 1
```

The current in terminal T/1 is measured to be 1 ampere.

```
(CURRENT T/2 (MEASUREMENT)) = 2
```

The current in terminal T/2 is measured to be 2 amperes.

```
(CURRENT T/3 (KIRCHOFFI N1) ()) = -3
```

The current in terminal T/3 is determined to be -3 amperes by using Kirchoff's current law (abbreviated KIRCHOFFI) to node N1 (which has only three terminals).

Kirchoff's voltage law states that if two voltages are known relative to a common point, the voltage between the other nodes can be computed:

```
(VOLTAGE (N1 N2)) = .5
(VOLTAGE (N2 N3)) = 6
(VOLTAGE (N1 N3) (KIRCHOFFV N1 N2 N3) ()) = 6.5
```

The voltage from node N1 to node N3 is calculated by Kirchoff's voltage law (abbreviated KIRCHOFFV) by summing the voltage from N1 to N2 with the voltage from N2 to N3.

Ohm's Law is equally easily illustrated:

```
(CURRENT R1) = 1
(VOLTAGE (N1 N2) (RESISTORI R1) (R1)) = 100
```

The voltage across the resistor is determined from the current through it by an instance of Ohm's Law (abbreviated RESISTORI). This propagation assumes the resistance of the resistor (100 ohms) has not changed and thus the assumption R1 is added to the ⟨assumptions⟩.

or,

```
(VOLTAGE (N1 N2)) = 100
(CURRENT R1 (RESISTORV R1) (R1)) = 1
```

These three kinds of calculations, or propagations, can be combined into a very simple propagation theory. First, Kirchoff's voltage law can be applied to every new voltage discovered in the circuit. Then for every node and component in the circuit, Kirchoff's current law can be applied. Finally, for every resistor having a newly discovered current in or voltage across its terminal, the Ohm's law can be applied to determine further propagations. If the study of the characteristics produces any new voltages or currents (via RESISTORV or RESISTORI propagations) the procedure is repeated.

This procedure is easily implemented, but strategies are needed to avoid making duplicate propagations. The simplest strategy is to only consider newly discovered values for making deductions and to not reapply the propagation of a component model to itself.

4.2.3.1 Component Experts

LOCAL has an expert for each type of device it models. These experts are based on the mathematical electrical engineering models, but are implemented with the propagation scheme outlined in the previous section. Each component of a given type must be modelled in the same way. If, for example, one transistor in the circuit were modelled differently from the others without there being some significant physical difference in the transistors, the modelling would have presumed the functionality of the overall circuit.

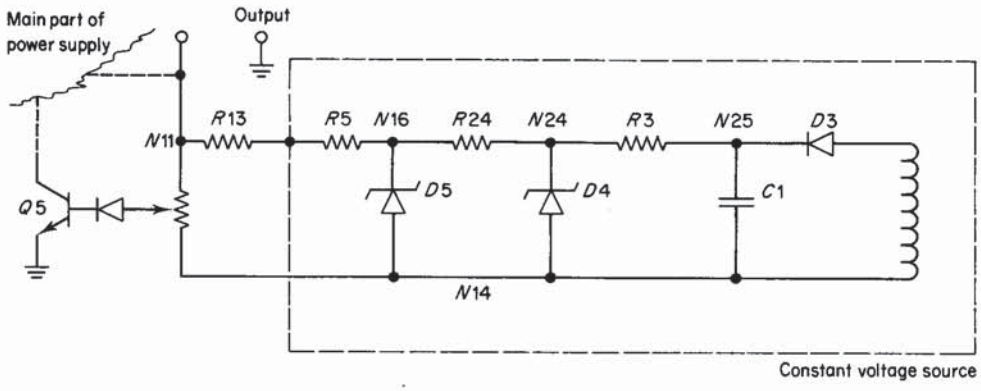

FIG. 4.3. Constant Voltage Source

Transistors, diodes, and zener diodes have discontinuous regions of operation. Each region is characterized by a fundamentally different behavior and the region usually needs to be identified before any values can be propagated. The diode is the simplest kind of semiconductor device; its only characteristic is that when it is reverse-biased (current can flow in only one direction through a diode), the current through it must be zero. From the zener diode we know that if the current through it is greater than some threshold, then the voltage across it must be at its breakdown voltage. Conversely if the voltage across the zener diode is at less than its breakdown voltage, then the current through the diode must be zero.

Suppose voltage measurements at the output and across D5 have just been made:

```
(VOLTAGE (N15 N14)  (MEASUREMENT)) =                    30
(VOLTAGE (N16 N14)  (MEASUREMENT)) =                    34
(VOLTAGE (N16 N15)) (KIRCHOFFV N16 N14 N15) ()) =       4
(CURRENT R5         (RESISTORV R5) (R5)) =              .003
(CURRENT D5         (ZENERV D5) (D5)) =                 0.
```

The voltage across the zener D5 is less than its breakdown, therefore the current through it must be zero.

```
(CURRENT R4         (KIRCHOFFI N16) (R5 D5)) =            .003
(VOLTAGE (N24 N16)  (RESISTORI R4) (R4 R5 D5)) =          7.18
(VOLTAGE (N24 N14)  (KIRCHOFFV N24 N16 N14) (R4 R5 D5)) = 41.18
(VOLTAGE (N24 N15)  (KIRCHOFFV N24 N16 N15) (R4 R5 D5)) = 11.18
(CURRENT D4         (ZENERV D4) (D4 R4 R5 D5)) =          0.
```

The voltage across zener D4 is less than its breakdown.

```
(CURRENT R3         (KIRCHOFFI N24) (D4 R4 R5 D5)) =            .003
(VOLTAGE (N24 N25)  (RESISTORI R3) (R3 D4 R4 R5 D5)) =          4.90
```

```
(VOLTAGE (N25 N14)  (KIRCHOFFV N25 N24 N14) (R3 D4 R4 R5 D5)) =   46.1
(VOLTAGE (N25 N16)  (KIRCHOFFV N25 N24 N16) (R3 D4 R4 R5 D5)) =   12.01
(VOLTAGE (N25 N15)  (KIRCHOFFV N25 N24 N15) (R3 D4 R4 D6 D5)) =   16.01.
```

We will have much more to say concerning the device experts later. First, though, the numeric values of the propagations must be examined in much greater detail since they are the source of a major complexity in the local propagator, LOCAL.

The device rules utilized in LOCAL are general. This propagator fulfils three roles in the articulate electronics expert. First, it determines the electrical consequences of measurements. Second, it predicts other measurements, assuming non-faulted components. And third, the predictions made by the propagator are used to interface to higher-order knowledge.

4.2.3.2 Range Arithmetic

All measurements in the circuit and all circuit parameters have some degree of error. The errors in circuit parameters originate from the fact that manufacturers cannot make perfect components and instead guarantee their specifications to within certain tolerances. (Typically the resistance of a resistor is only within 10 % of its specified value.) Similarly, the meter used to measure circuit quantities can only measure voltages and currents to certain accuracies (typically 2 % error). A further limitation is that the meter has a minimum range; SOPHIE's meter cannot accurately measure below .1 volt or below 1 microamp. These effects, though artificially introduced into SOPHIE, are representative of what the student would encounter with real circuits. Because the numerical computations performed by the device experts introduce truncation and roundoff errors, it makes more sense to propagate either values *and* their tolerances, or ranges of values. Consequently, the basic arithmetic operations performed by the device experts are modified to accommodate the tolerances associated with each measurement and circuit parameter. Inducing these problems reflects our concern for being able to coach the student as he works on real, physical circuits.

There are a variety of ways to introduce tolerances into the propagated quantities. SOPHIE III utilizes a very simple scheme, representing each quantity by a range indicating its extreme values. A range Q is described by $[Q_L, Q_H]$ which indicates $Q_L \leq Q \leq Q_H$. No probability distributions are maintained. Consider the example of a resistor expert. If we know the current through a resistor is between 2.9 amperes and 3.1 amperes ($I = [2.9, 3.1]$), and its resistance is 100 ohms with a 10% tolerance ($R = [90, 110]$), Ohm's Law tells us that the voltage across the resistor must be between 261 and 310 volts ($V = IR = [2.9, 3.1] \times [90, 110] = [261, 341]$).

By using range notation, a great deal of additional knowledge can be included in the device models. For example, knowledge that the current through a diode is always positive can be expressed by the range $[0, \infty]$. To prevent problems incurred by using ranges, two modifications must be introduced to the simple propagation scheme: first, propagations with very wide ranges must be stopped; and second, new values which are only marginally better (only marginally narrower ranges) than old ones must be ignored since they probably are the result of iteration.

4.2.4 Passive Troubleshooting

In addition to simple propagation, two other kinds of knowledge are required to troubleshoot. The first, which might be called "passive troubleshooting" is concerned with gathering information about the correctness of components from measurements. The second, "active troubleshooting" knowledge is needed to choose new measurements to make. In SOPHIE III the passive troubleshooting knowledge is incorporated into LOCAL as an extension of the propagator. The active troubleshooting is implemented with an independent articulate expert, and is parasitic on the entire electronics expert; it will be discussed later.

4.2.4.1 Coincidences, conflicts and corroborations

In this section we develop a simple but general strategy for deducing the correctness of circuit components based on information obtained by the propagation. We assume that the error in the circuit is that some component is not functioning according to its specifications, that the overall circuit was correctly designed, that the circuit contains no wiring errors, and that the circuit contains only one fault.

The discovery of a known value for a point for which we already know a predicted, propagated value is called a *coincidence.* When the two values are equal, we call the coincidence a *corroboration* — when they differ we call it a *conflict.* Coincidences provide information about the assumptions made in the propagation: Corroborations verify them and conflicts indicate at least one of them is in error. This simplistic notion must be substantially adapted before it can be useful in actual troubleshooting.

Its difficulty is that corroborations do not always imply that the components involved in the derivation are unfaulted. A component should only be considered unfaulted as a consequence of a corroboration if a fault in it would significantly modify its propagated values. This is not, in general, the case. The two most common exceptions occur when the circuit isn't manifesting a symptom or when a particular propagation does not depend significantly on one of its underlying assumptions.

Although a component is faulted, the overall system may still be functioning correctly. (The fault may only manifest itself under certain load conditions, for example.) Therefore a corroboration provides no information if the system is not manifesting a symptom under the specified external conditions (e.g., load and control settings).

The system may be exhibiting a symptom, but the way the faulted component causing the symptom is used in a particular propagation may not be significant. This occurs most commonly when a large quantity is added to a small quantity since the range of the large quantity completely swamps the contribution of the smaller quantity. A similar situation arises when the propagation multiplies an input value by zero. A propagation which uses the same component twice is also suspect since the two uses may cancel each other out. We call those assumptions which corroborations remove from suspicion primary assumptions, and those that do not, the secondary assumptions.

If all propagations consisted of simple arithmetic operations these rules would suffice to distinguish between primary and secondary assumptions. There is, however,

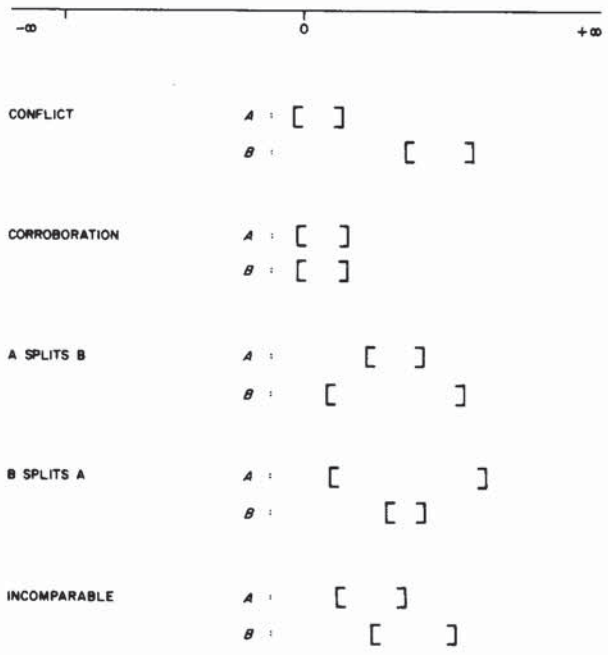

FIG. 4.4. Range Comparisons

one other kind of propagation which is based on a comparison of quantities. For instance, a transistor model may test the base-emitter voltage to check whether it is below a certain threshold and if so, conclude that the transistor must be off and therefore have zero collector current. If the base-emitter voltage is propagated to be some value below this threshold, the collector current being independently determined to be zero is not sufficient to corroborate the assumptions since the base voltage could be quite different (but below the threshold) without affecting the collector current. The assumptions underlying comparison to thresholds must, of necessity, always be secondary.

It is fairly easy to record and summarize the results of coincidences. A corroboration removes from suspicion all the primary assumptions. Similarly, a conflict removes from suspicion all assumptions not mentioned in the instigating propagation. The summary of all the coincidences is the union set of all the components removed from suspicion.

4.2.4.2 Consequences of ranges on coincidences

A comparison between two ranges can have one of five results (see Fig. 4.4): (1) values contradict, (2) values corroborate, (3) first value splits second, (4) second

value splits first, and (5) no comparison possible†.

Because the propagator does not use variables or perform algebra but only propagates numeric quantities it is not formally complete. For example, two measurements may produce propagations which coincide but which do not coincide at the original measurement points. If the propagator were complete, the first measurement would have resulted in a propagation for the second measurement point. This incompleteness requires that coincidences between two propagations also need to analyzed. This is much more complicated than the coincidences we have been considering since the assumptions of both propagations must be taken into account.

If either one of the propagations has no unverified assumptions, the coincidence can be handled as if it were between a propagated value and an actual measurement. However, if neither propagation is free from unverified assumptions the result depends on the intersection set of the unverified assumptions. If the intersection is empty, a conflict reduces the list of possible faults to the union set of the assumptions; a corroboration indicates that the common propagated value is correct, and can be considered as two separate corroborations between propagated and measured values.

The case of a nonempty intersection is more difficult. If the coincidence is a corroboration, a fault in the intersection could mean both propagations are incorrect yet corroboratory. (Even so, something can be said about the disjoint assumptions in the propagations, since if there were a fault in one of the disjoint primary assumptions the coincidence would have been a conflict; thus all the disjoint primary assumptions can be verified to be correct.) If the coincidence is a conflict, the list of possibly faulty components can be reduced to the union of the assumptions.

It is very tempting to remove from suspicion all components mentioned in the intersection, just as in the case of a conflict, and claim that correct propagations from a single (albeit incorrect) value must always corroborate each other; or, equivalently, that each point in the circuit has only two values associated with it — a correct value and a faulted value predicted by the propagator. Unfortunately, the claim is not valid.

4.2.4.3 Component Fault Modes

The theory of coincidences makes no presuppositions about the kinds of fault modes in which a component can be. But physical structure dictates that each component can only be faulted in a small number of characteristic ways. Since fault modes are not arbitrary and each fault mode has a distinct affect on propagations through the faulty component, a great deal more information about a component's faultedness can be obtained from propagations and coincidences. One very powerful strategy, called *conflict-trace*, examines the conflicting propagations to identify what

†The last alternative indicates that it is sometimes useful to propagate two independent values for the same quantity. The splitting possibilities can be intelligently dealt with. If the value for A splits the value for B, then if A is valid, B must be valid (though not vice-versa). For example, since A:[3 , 4] splits B:[0 , 10], the validity of A implies the validity of B. But if B were valid, A might be [7 , 8], still spliting B but contradicting with the original [3 , 4]. If A is not known to be valid, we must wait till it is proven before using this information. However, in a single-fault theory a very interesting deduction can still be made for this case: Every assumption of B which does not occur in A is corroborated. A split is just a kind of corroboration in which one of the propagations is much stronger than the other.

fault modes could have caused the particular conflict. Often, although a component may participate in the conflict, there is no fault mode in which it could cause the symptom, thus it must be blameless. Common fault modes for components are:

```
RESISTOR        open, shorted, high or low.
CAPACITOR       shorted or leaky.
DIODE           open or shorted.
ZENER DIODE     breakdownhigh or breakdownlow.
TRANSISTOR      beta-low, beta-high, all-junctions-open, ...
```

(The transistor has so many possible fault modes that it is impossible to choose succinct labels for each.)

Each component fault mode is characterized by a particular behavior. LOCAL does not utilize propagation models for the behavior of fault modes; instead, it has a collection of deduction strategies to detect what fault mode a component is in. Each of these deduction strategies is based on the common behavioral model of each device. There are four basic fault-detection strategies, each of which either is an integral part of the propagator, or examines the justifications for the propagations generated. *Excess* deductions result from LOCAL propagating into a component a voltage or current which drastically exceeds its rating, thereby indicating that it could be in a particular fault mode. *Inconsistent* deductions serve to rule out component fault modes whose behavior is inconsistent with the propagations. *Behavior* deductions are more subtle, and are based on the assumption that if a component is faulted and the overall circuit is manifesting a symptom, the faulty component must be manifesting a symptom. The *conflict-trace* strategy, unlike the previous three, does not participate in the propagations.

The conflict-trace inference strategy is the most complex of the four fault-mode detection strategies. It is a separate deduction mechanism independent of the propagator which is only invoked when a conflict or excess deduction determines some propagation to be high or low. Suppose that through a long propagation chain, we determine the current through a resistor and use Ohm's Law to predict the voltage across it. A subsequent measurement indicates this predicted voltage is too low. If the resistor is truly faulted, its resistance must be too high or be open. On the other hand, if it is not faulted the current supplied to it must have been too low; the process then recurses examining the component that was used to deduce that high current. (In either case, the resistor cannot be in the fault modes shorted or low.) For a single conflict, this technique will rarely verify more components, but it will always rule out faulted modes for each component involved.

One of the advantages of employing the fault modes is that more information can be gained from multiple conflicts: One conflict may indicate that if the resistor is faulted it must be high or open while another indicates that if the resistor is faulted it must be low or shorted. The combination of these two conflicts indicate the resistor is unfaulted.

In any justification each component expert has recorded how each component contributes to the propagation. This information can be used to determine the fault mode of the component, or which erroneous input propagations (if any) could cause the observed symptoms. Consider the example of a high voltage deduced by applying Ohm's Law. The resistor expert will have recorded the justification of the voltage

with an annotation that it used a current to deduce a voltage by Ohm's Law. The necessary conflict-trace rule is: "All current-to-voltage propagations, produced by a resistor, that are too high indicate either that the resistor is shorted or low, or that the resistor's input current is too high."

The conflict-trace deductions are performed by a separate propagator which, instead of propagating ranges, through the circuit topology, propagates the tokens "high" and "low" through the justification of the problematic propagation. The tokens refer to whether the propagation is higher or lower (the ranges do not overlap) than was actually measured. The potential difficulty is that, if a component occurs more than once, it is not easy to tell which contribution dominates. To avoid this problem, LOCAL performs the entire conflict-trace deduction as one unit, identifying which fault modes on which components could have caused the observed symptoms and localizing the fault to the union of these. Thus, if the same resistor occurs twice in the same conflict, one explained by high or open and the other explained by low or shorted, no deduction about that resistor will be made other than it is under suspicion.

4.2.4.4 The Model for a Diode

Although every component of a given type behaves in the same basic way, the fine details of its behavior are controlled by its parameters. These parameters are determined by the manufacturer, not by the component's use in the circuit. For example, every resistor obeys Ohm's Law, but with varying resistance. The model for the diode is specified by eight parameters, each parameter is listed with its value for D6:

I_{MIN} : the maximum allowable reverse current flow, —1 microampere.
I_{MAX} : the maximum allowable forward current flow, 1 ampere.
V_{MIN} : the minimum voltage across the diode, —50 volts.
V_{MAX} : the maximum voltage across the diode, .8 volts.
I_{OFF} : defines the diode OFF state, 1 microampere.
I_{ON} : defines the diode ON state, 2 microamperes.
V_{OFF} : defines the diode OFF state, .3 volts.
V_{ON} : defines the diode ON state, .45 volts.

These parameters are used by the diode expert to propagate new currents and voltages. If a newly discovered voltage is less than the threshold required to turn on the diode, the diode cannot be conducting very much current; on the other hand, if the new voltage indicates the diode is on, the diode must be conducting a significant amount of current:

A new voltage $V = [V_L, V_H]$ causes the propagation: If $V_H \leq V_{OFF}$, propagate the range $I = [-\infty, I_{ON}]$, otherwise if $V_L \geq V_{OFF}$ propagate the range $I = [I_{ON}, +\infty]$.

FIG. 4.5. Diode Voltage Parameters

(Note that we could not even express this rule without ranges.)

The propagations which result from a new current are analogous:

A new current $I = [I_L, I_H]$ causes the propagation: If $I_H \leq I_{OFF}$, propagate the range $V = [-\infty, V_{OFF}]$, otherwise if $I_L \geq I_{ON}$ propagate the range $V = [V_{ON}, +\infty]$.

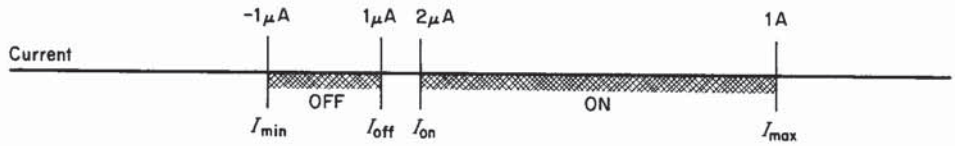

FIG. 4.6. Diode Current Parameters

Components have certain maximum limits which cannot be exceeded without destruction. These limits have three effects: They modify the above propagations, allow the model to explicitly detect faulty behavior, and allow some ranges to be propagated before any measurement is made. Wherever $+\infty$ or $-\infty$ appears in the above rules, they should therefore be replaced with the appropriate maximum or minimum parameter. Potentially faulty behavior can be detected by comparing the new propagation with the diode's maximum specifications:

If $I_H \leq I_{MIN}$ the diode must be shorted or I must be too low (I has a lower value than it should have). If $I_L \geq I_{MAX}$ the diode must be shorted or I is too high. If $V_L \geq V_{MAX}$ the diode must be open or V = high.

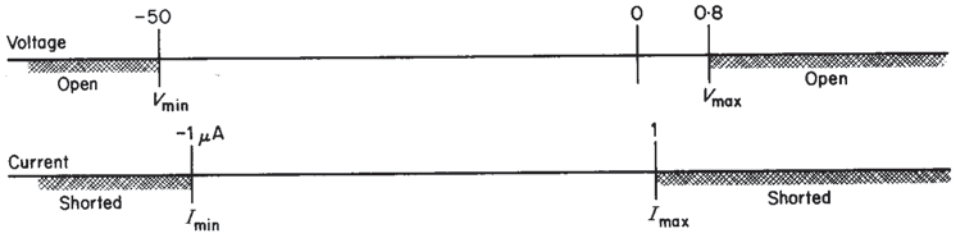

FIG. 4.7. Diode Ratings

The expressions "I = high" and "V = high" indicate that a conflict-trace model must be applied to whichever component deduced the voltage or current that triggered the diode expert. The disjunction in the rule "If $V_L \geq V_{MAX}$ the diode must be open or V = high" states that one of "diode is open" or "V = high" must be true. Since "V = high" indicates the propagated value for the voltage is higher than it should be, this rule asserts that either the diode is open or one of the assumptions underlying V is incorrect. For instance, if the voltage across the diode was simply measured, there would be no underlying assumptions and the diode must perforce be open.

In the case of a diode, the maximum ranges can be included in the propagation database before any measurements are made.

Certain propagations are inconsistent with the diode being in a particular fault mode. If the voltage across a diode is greater than zero, it cannot be shorted since the definition of shorted is that the voltage across the diode is zero. Similarly if the current through a diode is greater than zero it cannot be open.

If $V_L \geq .1$ the diode cannot be shorted. If $I_L \geq I_{OFF}$, the diode cannot be open.

The behavioral deductions from a propagation are based on the assumption that the circuit contains only one fault and that it is currently manifesting a symptom. If the voltage across the diode is less than some maximum, the diode cannot be open since it cannot be causing a symptom. If the current through the diode is less than enough to warrant it being considered on, it cannot be shorted since if it were shorted and causing a symptom it would be conducting a significant amount of current.

If $V_H \leq V_{MAX}$ then the diode cannot be open. If $I_H \leq I_{ON}$ then the diode cannot be shorted.

When the value propagated by the diode leads to a later conflict, the fact that the propagation was high or low can be used to determine the fault modes in which the diode could be. The diode model can make six different propagations: (1) from a new voltage deduce that the diode is on and propagate the corresponding current, (2) from a new voltage, deduce that the diode is off and propagate the corresponding current, (3) from a new current, deduce that the diode is off and propagate the corresponding voltage, (4) from a new current, deduce that the diode is on and propagate the corresponding voltage, (5) add extreme maximum and minimum voltages to the propagation database before any measurements are performed, (6) add extreme initial maximum and minimum currents.

The conflict-trace rules associated with each of these six propagations are very complex, and therefore we have included only the one; for the first case (1). This conflict-trace rule will presume all the preceding diode rules. If the current propagated by the diode model is high (i.e., the predicted is higher than the measured), the diode must be shorted since the current flowing through it must be less than I_{MIN}. No erroneous value of the voltage propagated into the diode can cause this symptom since there is no valid region of operation in which the diode current is less than I_{MIN} (I_{MIN} is negative, and represents the maximum reverse current flow).

On the other hand, if the propagated current flowing through the diode is low (i.e., the predicted is lower than the measured), the diode must be shorted, or the voltage across the diode is high (indicating that the diode is off rather than it being on).

For the propagation V -> I diode off: If I = high, diode must be shorted, and symptom cannot be caused by V. If I = low, diode must be shorted, or V = high.

The diode being open never arises since a voltage low enough to indicate the diode is off will also trigger the behavioral deduction that the diode cannot be open.

4.2.5 Coaching and Active Troubleshooting

The articulate electronics expert makes predictions (propagations) from student measurements deduces their effect on the faultedness of circuit components. These deductions form the basis from which the coach monitors the student's troubleshooting progress. LOCAL might discover that the student consistently made redundant measurements indicating his lack of understanding about how a transistor operates. The coach might ask the student, following some measurement, whether some component was faulted and so discover whether the student understood corroborations. Coaching interactions such as this are supported by the deductions of the electronics expert. However, the skill of troubleshooting is not as much concerned with the consequences of measurements as it is with the choice of which measurement to make. SOPHIE III's coach must be able to comment on the quality of the student's measurements, as well as to suggest new ones. To this end, SOPHIE III employs a second expert, the articulate troubleshooting expert. Although the troubleshooting

expert and the electronics expert construct a database of the deductions necessary to support coaching interactions, the coaching expert was never fully implemented. Instead SOPHIE III employed a skeletal coach which served mainly to suggest the kinds of coaching interactions that might be possible.

The skeletal coach of SOPHIE III always answers the student's questions, but can only initiate interactions just after the student states his choice of measurement, and again just after he obtains its results. Some measurements automatically can be determined to be uninteresting since they involve a coincidence with a narrow numeric range, based on verified assumptions. In such cases the coach will point out to the student that the measurement would be uninteresting, and the student can then inquire about the details. On the other hand, if the measurement is interesting, it is guaranteed to propagate new voltages and currents which may or may not remove more components from suspicion.

In order to determine the quality of a measurement, SOPHIE III requires a theory of troubleshooting. The goal of troubleshooting is to remove as many components from suspicion as efficiently as possible. Thus, the basic dimension by which to judge the quality of a student's measurement is the number of new components it removes from suspicion (easily determined by the articulate electronics expert). The coach should not critique a student measurement though, solely on this basis; it should only critique if it knows of some more informative measurement.

Measurements are evaluated by hypothesizing their coincidences with propagations. The quality (score) of the potential measurement is the *expected* number of components that it would remove from suspicion. The expected value is the sum of two products: — the number of components that would be verified by a corroboration times the probability of a corroboration, plus the number of components that would be verified by a conflict times the probability of a conflict. The student is therefore critiqued only if he makes a measurement with a score significantly lower than optimum, and is complimented if he makes a near-optimal measurement.

A surprising feature of this coaching strategy is that the student may be critiqued even if he makes a measurement that verifies more components than the optimal measurement would. For example, he might make one measurement which localizes the fault to forty components, then make a second measurement (which causes a conflict) which localizes the fault to one component. If a corroboration would have verified only one component, the choice of measurement is a very poor one (there was only a slim probability that the component was at fault). The student could be critiqued for his successful measurement — it was pure luck that he found the fault, since the expected value of that measurement was low.

The troubleshooting expert can also be used to make measurement suggestions when the student asks for them. It is a simple matter to identify the measurement with the best score. The explanation for the proposed measurement is simply a description of what components the coincidence could potentially verify and why. This measurement-suggestion mechanism can be used as a troubleshooter in its own right, with the student merely making the measurements, watching the expert explain the reason for the measurement, and explaining its consequences.

The coaching strategy presented thus far is not very good. This simple coach presumes to know all there is to know about electronics and presumes to have

a totally accurate model of the student at all times. Unfortunately, because the electronics expert is far from exhaustive, a student who appears "lucky" may just, in fact, be smarter than the computer-based expert. A more conservative critiquing strategy would be to only critique the measurement if the hypothesized outcome matched. A less devastating, but nonetheless serious, problem is that during the early phases of troubleshooting the propagator may not know of hypothetical measurements that are close to the theoretical optimum. Before the first conflict the list of suspect components contains approximately all the components in the circuit. A near-optimal measurement, which should coincide with a propagation that uses half the components, is unlikely that such propagations would occur. On the other hand, once a single conflict is found, optimal measurements can be found which coincide with subpropagations (e.g., intermediate propagations halfway between the original measurements and the site of a conflict) of the conflicting propagations.

Aside from the fact that the expert does not make all possible inferences nor construct an adequate model of the student's capabilities, there are a variety of better coaching strategies available. For example, the coach might question the student when the expert discovers some important result in order to check whether the student has made the same deduction. Whenever the student makes a measurement which appears suboptimal, the coach could begin questioning the student about which components were faulted, and so diagnose which troubleshooting strategies the student does not understand.

4.2.6 The Need for Higher-Level Knowledge

There are still serious difficulties in tracking the troubleshooting of the student; this is particularly evident early in a debugging scenario. The first measurement, one even a neophyte debugger would perform, is to measure the output. This is probably the single most informative measurement he can make, LOCAL can do nothing with it. Only after the student has made a number of measurements can enough values be propagated to derive some useful coincidences. In this opening sequence, the troubleshooter, no matter what is skill level, is basing his measurements on a different theory than the one LOCAL is using.

The expert is missing all the the attributes of "understanding how the circuit works." It does not understand the causality or the teleology of the circuit; in fact it does not even have a hierarchical description of the functional components. It employs a myopic local view of the circuit and thus fails to encompass the global mechanism through which the circuit functions. Its myopic view is only appropriate when the fault has been isolated to some module or group of components; it is entirely inappropriate for making initial measurements.

Our current state of knowledge about causal and teleological reasoning is of insufficient depth to permit their inclusion in the expert in a general way. Instead, we summarize the results of such reasoning and use these to make deductions. This new knowledge, unlike the propagator, is circuit-specific and must be changed for every new circuit SOPHIE III encounters. There are two principles we employ to encode this circuit-specific knowledge. First, we impose a strict discipline upon its form, making addition, modification and explanation of higher-level knowledge easier. Second, the

propagator and its underlying mechanisms for handling assumptions and fault modes are used as a way of simplifying the deductions that the higher-level knowledge needs to make. Consider a simple example, suppose we had the rule "if output voltage is low, the regulator is shorted." The output voltage can be determined in a number of ways, one of which is by measuring the current in the load and using Ohm's Law to determine the voltage across it. The rule tells us that if the voltage is directly measured to be low, the regulator must be shorted. If the current is measured to be low, then either the regulator is shorted or the load is open or high. By utilizing the propagator and using the assumptions of the propagation (in this example, that the resistance of the load hasn't changed) to modify the consequences of the higher-order rules we need only one rule to handle both of these circumstances.

In addition to losing generality, we also lose some capacity to make intelligent explanations since the higher-level rules are made by the person who constructs the rules and not by LOCAL. Fortunately, there are four effects that mitigate this loss. First, the rules are relatively primitive, and thus any particular troubleshooting deduction will use a sequence of them, thus admitting a kind of deductive argument but with prestored explanation for each step in the sequence. Second, the rigid discipline on the form of the higher-order rules makes it much easier to impose a coherent structure on the rules. Third, the use of the propagator limits the necessary rules to a small essential core. Fourth, whatever contributions LOCAL makes to each deduction can be completely explained in terms of general electronic laws.

Although a circuit is constructed from components, it is more easily understood as consisting of a small number of interacting modules which in turn, consist of other modules. All of the circuit-specific knowldge is organized around these modules and their behaviors. The hierarchy of modules of a circuit are represented by a tree, the root of which is the circuit itself, the nodes of which are its modules and submodules, and the leaves of which are its components. Reasoning about module behaviors is of three types. First, a module's behavior is determined by the voltages and currents on module ports noted by LOCAL. Second, its behavior has implications for its submodules and parent modules. Third, the behavior of one module can causally affect the behavior of a neighboring one. The first kind of reasoning is implemented with a quantitative-qualitative map and a collection of rules for maping qualitative circuit values to module behaviors. The second is implemented by an inheritance mechanism called the behavior-tree. And the third is handled by a modification of the same rule-based system which detects module behaviors.

4.2.6.1 Module modes

Just like components, modules can be considered to have behavioral modes. One of the advantages of employing module behaviors is that they refer to "natural" internal states. These states must always be deducible by considering the module as a black box and looking at its input-output behavior. The identification of the "natural" module modes is a difficult process. One major difficulty is that modules usually have four or more terminals, while components usually have only two. To circumvent this difficulty, the mode of a module is directly identified with its behavior on one chosen port. For example, the mode of the constant current

source is determined by its output voltage and current.† The modes used in SOPHIE III were chosen on the basis of simplicity, and the majority of them have only one or two (high and low) faulted modes. The exact choice of modes does not matter much, but it is critical that the different reasoning strategies which utilize modes agree upon the behavioral meaning of a module's modes, otherwise SOPHIE III will encounter irreconcilable contradictions.

A module is operating in a faulted mode if it is exhibiting the symptom specified by the semantics of the mode description and contains a fault causing that symptom. A module which contains no faults is by definition operating in a valid mode. A module can be exhibiting the symptoms of a faulted mode, yet be in a valid mode because some component external to the module faults‡. The possible fault modes need not be disjoint. Also the same component fault can cause two different behaviors depending on the precise shift in parameter value and the control settings.

Knowing a module's behavior provides information about its parent and constituent modules behaviors. These necessary deductions are not easy; a given constituent behavior can produce different behaviors in the parent, and even the same component fault mode may cause different module behaviors. The reasons for this difficulty are that component fault modes are not totally specified (e.g., R8 being high can mean R8 has a resistance between 3 and ∞ ohms) and that different control settings can cause the same component fault to manifest slightly different circuit symptoms. The knowledge engineer must construct a circuit-specific knowledge structure which indicates how each module's behavior contributes to the behaviors of its parent. We call this structure the behavior-tree (although it is really a lattice). Every node in the behavior-tree corresponds to a behavior mode of a module. The downward edges (see Fig. 4.8) from a node correspond to the behaviors of its submodules that could cause the node's own behavior; the upward edges from a node correspond to the parent behaviors that the node can cause.

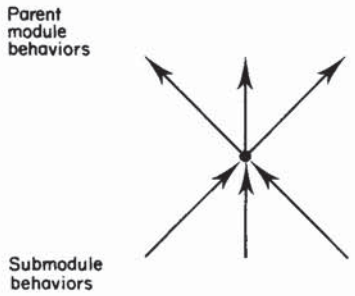

FIG. 4.8. A Node in the Behavior Tree

The terminals of the behavior tree are the fault modes of the components.

†Because we have no principled way of choosing the behaviors, some care must be taken since the extreme case is a one-to-one mapping of faulty module behaviors to constituent component faults.

‡This could be more general. We should include the possibility that a module may be faulted but not manifesting any symptom.

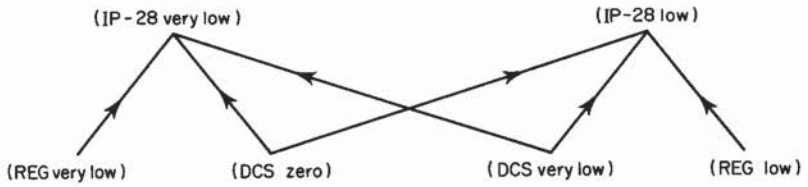

FIG. 4.9. Part of the IP-28 Behavior-Tree

A node in the behavior tree is completely specified by the module name and the mode it is in which can be abbreviated (<module> <mode>). Fig. 4.9 describes a small part of the IP-28 behavior-tree.

The behavior (IP-28 very-low) can be caused by (REG very-low), (DCS zero), or (DCS very-low). The behavior (IP-28 low) can be caused by (DCS zero), (DCS very-low), or (REG low). Note that (DCS zero) and (DCS very-low) can cause both (IP-28 very-low) or (IP-28 low).

The following sections will describe an inference strategy which uses the behavior-tree to deduce new module behaviors. The knowledge engineer need not be aware of this mechanism, but he must obey certain constraints when constructing the behavior-tree if this mechanism is going to be effective. The basic constraint is that the behavior-tree must be complete, so that anything not mentioned in the behavior-tree can be assumed to be impossible or false. The behavior tree must:

(1) Include every possible mode a module can be in (faulty and valid).

(2) Show every faulty behavior as causing at least one other faulty behavior (except for the overall module).

(3) Include all possible ways in which a module can affect a parent.

(4) Include every fault of every component†.

These constraints are important since they, in effect, imply that the behavior-tree and its module behaviors have a distinct unique meaning.

4.2.6.2 Detecting a module's behavioral mode

All of LOCAL's deductions are constantly monitored to see if some module's behavior is yet known. This monitoring process consists of two stages. The first

†Some faults immediately cause others, so these are left out of the tree by the single fault presuppostion.

detects interesting values discovered by LOCAL and adds these to a qualitative database, and the second uses this database to make assertions about module behaviors.

The propagator watches module ports and when voltages and currents are discovered it adds the qualitative values, corresponding to their ranges, to the database. This correspondence is specified by a translation table indicating what numeric values correspond to which qualitative value. Since the propagator uses ranges, some values may overlap more than one qualitative value. To handle this case, the electronics expert computes the intersection of the results that the each of the overlapping values individually.

As an example, consider the watch-point at the output of the IP-28 voltage reference (the voltage between nodes N11 and N14). The translation table is (high $\geq$ 31.0 $\geq$ eq $\geq$ 27.0 $\geq$ low). The numbers in the table delineate the different qualitative ranges. If the voltage is above 31.0 volts (VOLTAGE (N11 N14) high) will be added to the qualitative data base. If the voltage lies between 31 and 27, (VOLTAGE (N11 N14) eq) is asserted. The voltage [30 , 32] overlaps high and eq, so this voltage results in the assertion (VOLTAGE (N11 N14) (OR high eq)). This disjunction is handled by first deducing all of the consequences of (VOLTAGE (N11 N14) high) and then all of the consequences of (VOLTAGE (N11 N14) eq), and only permanently recording those consequences which are common to both. SOPHIE III employs 12 other watch-points to model the IP-28.

The detection rules which trigger on the qualitative database assertions need not pinpoint the exact behavioral mode of the module. They can determine that the module is in one of a set of possible modes (a localization), or determine that the module cannot be in some mode (an elimination). The mechanism for localizing and eliminating module behaviors is analogous to those used to deal with the fault modes of components: A deduction cannot eliminate a faulty behavior if the module contains a component about which the deduction makes an assumption.

The detection rules can also trigger on the states of components. Semiconductor components can have many states, each modelled by a distinctly different characteristic behavior. The component experts add their state into the qualitative database.

Module behaviors are not disjoint. Thus, it is not only necessary to know all the possible behaviors of a module, but also which behaviors are inconsistent with each other. For example, the regulator behavior low is inconsistent with behavior high, but not with behavior very-low. As with component fault modes, all localizations are transformed to the equivalent eliminations and recorded as such. Components are treated as degenerate modules whose behaviors correspond to the modes of its single constituents.

The rules for the variable voltage reference of the IP-28 are:

```
(VOLTAGE (N11 N14) HIGH) -> (LOCALIZE VARIABLE/REF high ok)
(VOLTAGE (N11 N14) EQ) ->  (LOCALIZE VARIABLE/REF ok)
(VOLTAGE (N11 N14) LOW) -> (LOCALIZE VARIABLE/REF low)
```

Ok is added in the first case because the IP-28 voltage-reference is designed to actively correct a low voltage on its output terminal, but not a high voltage. Thus if a low voltage appears, the voltage reference must be faulted, but if a high voltage appears some external module could also be supplying a voltage to its port. This latter

possibility never arises in normal operation, but can in faulty operation.

4.2.6.3 Reasoning with the Behavior-Tree

The behavior-tree is used by SOPHIE III to make inferences about module behaviors. The inference mechanism operates solely on a database which records the eliminated behavioral modes of modules. All localizations are converted to their equivalent eliminations so we need only consider inferences resulting from an elimination. The underlying idea is that when a faulty behavior is eliminated it can no longer cause, or be caused by any other behaviors. Eliminating a node involves searching up and down the behavior-tree for other behaviors to eliminate. If a faulty node is eliminated, the nodes below it can no longer be the cause; each of these lower nodes is examined to see whether it can still cause any other behavior, and if not, it too is eliminated. Analogously, if a mode is eliminated it can no longer cause any of its parents, so each parent behavior it can cause is examined to see whether it still can be caused by another behavior, and if not it too can be eliminated. One elimination can thus recursively generate many more. If all possible faulty behaviors of a module are eliminated, the module cannot be faulted.

Elimination and localization of valid behaviors are handled similarly, although more simply. While a faulty behavior can be caused by a faulty behavior in any one of the submodules, a valid behavior only results if all of the submodules are behaving validly. Thus the elimination of a valid node has no direct consequences on the nodes below it since the elimination of any one of these subnodes alone would be sufficient to eliminate the node itself. On the other hand, when a valid mode is eliminated its parent node cannot happen unless it could also be caused by an alternative valid mode of the module. If all possible valid behaviors are eliminated, the module must be faulted.

The remaining kinds of inferences are concerned with handling deductions about faultedness in general. If a module is determined to contain a fault, all of its parent modules are, by definition, also faulted. Since a particular faulty mode need not be known, this is achieved by eliminating all of the valid modes of the parent modules. Under the single-fault assumption and the fact that modules do not overlap, any sibling modules are automatically unfaulted. These mechanisms are sufficient to handle all the necessary deductions about behaviors. Consider the rule that if a module is unfaulted, all of its submodules are unfaulted. This is a simple consequence of the fact that all of its faulty behaviors have been eliminated and therefore its submodules have no possible faulty behaviors either. Similarly, if a parent module is known to contain a fault, and only one of its siblings can still contain a fault, that sibling contains the fault.

The propagations which cause the original elimination may have underlying assumptions which modify the consequent eliminations and localizations. These assumptions must be carried along during the recursive eliminations and localizations, continuously enforcing the restriction that an elimination of a faulty mode cannot depend on an assumption about the module in question. Eliminations of valid modes can, of course, depend on the module itself. When an illegal elimination of a faulty behavior occurs, the recursive eliminations are still attempted. The underlying

assumptions may refer to only some of the components of the module, in which case the elimination will still lead to deductions about that part of the module which does not overlap with the underlying assumptions of the original elimination.

4.2.6.4 Non-hierarchical interactions

By far the majority of SOPHIE III's circuit-specific rules detect module modes. The behavior-tree, unfortunately, only captures hierarchical module interactions, and thus fails to capture many of the causal interactions between same-level modules. The state-to-state rules used by SOPHIE III, which are used to handle this latter case, can be divided into three categories. The first kind of rule, and also the most common, deduces that a module's neighbours are unfaulted when the module itself is unfaulted. The second identifies the causes for why a module behaves sympomatically, even though unfaulted. The third is a particularly inelegant type of rule that eliminates faulted modes of modules which are topologically distant from the watchpoint. We have tried to avoid rules of this last sort, but SOPHIE III employs three of them to capture the global interactions of non-neighbouring modules†.

The first kind of state-to-state rule concerning correct behavior, concentrates on determining neighbors to be correct. Often, when a module is discovered to be unfaulted and behaving normally (as distinguished from being unfaulted but behaving symptomatically) its "suppliers" and "consumers" can also be determined to be behaving normally, and hence to be unfaulted. Consider an example from the IP-28. The constant-voltage reference of this power-supply (see Fig. 4.3) consists of a raw DC voltage which is first regulated to 56 volts and then regulated down to 36 volts. The only supplier of power to the 36-volt regulator is the 56-volt regulator, and the only consumer of the 56 volt output is the 36-volt regulator. If voltage of the second reference is 36 volts, it is operating normally and unfaulted, and the 56-volt regulator must be unfaulted and operating normally. So the module-detection rule,

```
(VOLTAGE (N11 N16) EQ) -> (LOCALIZE 36V/REG operating-ok)
```

triggers the state-to-state rule,

```
(36V/REG operating-ok) -> (LOCALIZE 56V/REG operating-ok),
```

which, in turn, triggers the state-to-state rule,

```
(56V/REG operating-ok) -> (LOCALIZE LQ/DCS operating-ok).
```

(The LQ/DCS is the source of raw unregulated DC for the reference.)

The second type of state-to-state rule is somewhat analogous except that it is concerned with the faulty behavior that can be caused by neighbouring *faulty* behaviors. Consider the voltage-reference example again. If the output of the 36-volt regulator is low it is either faulted, being supplied with a bad signal (the 56-volt regulator is low), or being presented a bad load (the output filter of the voltage reference is shorted). Earlier we presented the mode-detection rule as:

† To some extent, all of the rules are inelegant. If we were going to allow circuit-specific knowledge it would have been more parsimonious to construct a qualitative causal model of the circuit which could be used to derive the state-to-state rules. Unfortunately we were unable to do so. It should be noted, however, that the reason so many rules are required (39, including the detection rules) is that we needed to track a student's potentially pathological measurements. If we were solely concerned with optimal troubleshooting, a dozen of the detection rules would have been sufficient.

```
(VOLTAGE (N11 N16) low) -> (LOCALIZE 36V/REG low ok),
```
but to facilitate the propagation rules we write it as,

```
(VOLTAGE (N11 N16) low) -> (BEHAVIOR 36V/REG low)
```
The effect is the same, except that the module-mode propagation rule can now trigger on the BEHAVIOR assertions:

```
(BEHAVIOR 36V/REG low) -> (BEHAVIOR 56V/REG low)
...
```
The effects of these rules can propagate through neighbouring modules, .e.g.,

```
(BEHAVIOR 56V/REG low) -> (BEHAVIOR LQ/DCS low).
```
These type of rules are powerful because they allow us to conclude, after all the possible BEHAVIOR rules have applied, that any module not mentioned in a behavior assertion must be unfaulted. This follows from our requirement that every possible cause for a module's faulty behavior must be included in the rule set. In effect, we have a disjunctive localize, e.g., the above example becomes a (OR (LOCALIZE 56V/REG low) (LOCALIZE 36V/REG low) (LOCALIZE LQ/DCS zero) (LOCALIZE VC/POT low) ...).

The third and last type of state-to-state rule is an exception rule to handle the cases which do not fit either of the previous two patterns. They are all of the form:

```
<measurement> -> (ELIMINATE <module> <mode>).
```
This pattern is the same as that for the module mode detection rules, but unlike those rules, the measurement takes place away from an input-output port of the module. For example, one of the three such rules for the IP-28 is:

```
(VOLTAGE (N11 N14) EQ) -> (ELIMINATE Q5 sh/op/op).
```
(The transistor mode sh/op/op is the collector open, and the base shorted to the emitter.) There are two components between Q5 (see Fig. 4.10) and either of nodes N11 and N14. Unfortunately, this rather inelegant rule is needed. This particular fault mode for Q5 makes the voltage regulator inoperative, and also shorts the output of the voltage reference to the output of the overall power-supply. The voltage reference is very weak, so the main power-supply will dominate. Thus, if the output of the voltage-reference is normal, Q5 must not be shorted in this way. (If it were, it would not be voltage regulating, causing the voltage across the reference to be high.) This is a very subtle inference which would not be generally possible without a rather sophisticated causal and teological model of the IP-28.

4.2.6.5 Troubleshooting with circuit-specific knowledge

Although the inferences made by the circuit-specific knowledge are more powerful than LOCAL's, their structure is simpler. All circuit-specific knowledge can be reduced to a collection of if-then rules making it easy to calculate the consequences of hypothetical measurements. The same techniques for using the general knowledge to score measurements apply to the circuit-specific knowledge. Recall that the best strategy is to identify a measurement with the maximum expected information value. The scores of each of the measurements considered by the general troubleshooter are augmented by the information gained from ensuing circuit-specific deductions. In

addition, measurements at each of the watch-points in the circuit's translation table are considered and their expected value computed. Thus SOPHIE III can troubleshoot the instrument by itself. Early in the troubleshooting the measurements at module ports will have higher scores (at the watch-points); later in the troubleshooting the scores for hypothetical coincidences with propagations will be higher.

The troubleshooting coaching strategy remains unchanged. However, the computation of expected values by the troubleshooting expert must be extended further than just the addition of circuit-specific knowledge seems to suggest. Consider the case where the student makes a measurement which is neither at a watch-point, nor at an earlier propagation. Such measurements can be useful when they propagate into a watch-point (which happens fairly often because the student's opinion of the module boundaries differs from the expert's). Because only the expected value is of any use, SOPHIE III computes an approximation to the expected value of the student's measurement by hypothesizing different outcomes. Since SOPHIE III does not use this expensive strategy to evaluate the other hypothetical measurements, its expected for hypothetical measurements will always be lower than they actually are. It does not consider the influence of propagations into multiple watchpoints, or into a watchpoint but with underlying assumptions unless computing the expected value of a student's actual measurement (in which case the computational cost is justified). Therefore, the possibility exists that SOPHIE III may make slightly suboptimal measurements. Fortunately, the result is that the coach criticizes the student less, rather than more, than it actually should.

4.2.7 Problems With SOPHIE III's Electronics Expert

The electronics expert, although extremely powerful, suffers from several limitations. These limitations take two forms: those resulting from SOPHIE III's incompleteness and those resulting from SOPHIE II's inconsistency.

Contradictory deductions (inconsistency) are the consequence of making presuppositions about the circuit behavior which don't necessarily hold for the system or fault in question. There are three primary sources responsible for contradictions: the presupposition of a single fault, the presupposition that all fault modes are known, and the presupposition that a symptom must be caused by a component being in a fault mode. Fortunately, there are techniques which could be employed in SOPHIE to handle the situations that violate these presuppositions.

The single-fault presupposition is the most pervasive. Almost every deduction employed by SOPHIE III relies on it. Without it a corroboration cannot logically be used to verify the underlying components nor can a conflict be used to verify the components that aren't mentioned in the underlying assumptions that lead to the conflict. Much of the reasoning mechanism of the behavior-tree is no longer valid.

The case of the single-fault presupposition is indicative of the other two. Without making these presuppositions, powerful troubleshooting cannot occur; adhering to them too stringently can lead to irreconcilable contradictions. These presuppostions are thus very good heuristics for troubleshooting and should be assumed to hold only until evidence is discovered to the contrary. In this light, they should be treated as *defeasible assumptions* subject to explicit reasoning, much like the component

assumptions of LOCAL. They are, in essence, a deeper level of assumptions, examined only when all the component assumptions have been eliminated. Every deduction made by LOCAL which depends on the single-fault presupposition should explicitly include this fact as an underlying assumption of that deduction. Unlike the component assumptions, these *presuppositional assumptions* influence the inference mechanism making the deductions. Although when we built SOPHIE III we had some ideas of how to handle this situation as a special case, it is the pioneering work of John Doyle [79] that suggests a general framework for defeating such assumptions, to elegantly alter the inference mechanism, and to efficiently reinstate only those previous deductions that are still valid. Within this extended framework, the troubleshoooting module, which mainly works on assumptions and justifications produced by the expert reasoning engine, would become just a special case of a more powerful reasoner that was capable of "deliberating" over its own actions and theories.

We have been discussing one alternative to the single-fault case, that of multiple independent faults. A more common case is that of multiple consequential faults. The single-fault heuristic often works well for such multiple faults. Frequently, the component that originally faulted will no longer be manifesting a symptom since the consequential fault has effectively made the originally faulted component appear unfaulted by isolating it.

The second presupposition, that SOPHIE III presumes all possible faults are known, is again good heuristic, but again does not always hold. Topological faults such as a short between two nodes which do not share a common component (e.g., two geometrically adjacent, but topologically distant, components which have been physically bent so that their leads touch) or an open node which disconnects two or more components from two or more others are such violations. A more subtle violation occurs with component fault modes, since it is hard to hypothesize all the ways in which a component can fault. Obviously a diode cannot turn into a battery, but under extremely rare conditions it can turn into a resistor that is neither open or shorted. This can cause problems for the behavior-tree mechanism, since the module may end up in a mode which was not hypothesized by the knowledge engineer. Fortunately, this is not usually a problem for SOPHIE III, since if no component is manifesting a symptom, the overall system cannot be manifesting a symptom. Nevertheless, just as with the single-fault assumption, this one should also be included as an assumption underlying every deduction involving an individual component or module.

The third important presupposition, that a circuit symptom is a direct consequence of some component behaving symptomatically, is only true for circuits which do not have some kind of "memory." Suppose a device had a circuit breaker on its input which blew every time the power supply was plugged in. The power supply is manifesting a symptom, but every component is functioning correctly: LOCAL would encounter an irreconcilable contradiction. The problem is, of course, that the circuit breaker "remembers" that some component was behaving symptomatically, even though the component might not be doing so at present. This type of fault is notoriously hard to find since the troubleshooter does not get the opportunity to see the faulted component manifest its symptom. Note the strong similarity between

this situation and that of consequential faults: In both cases the source fault must be discovered without the propagations of LOCAL.

5 Conclusions and Theoretical Observations

There are many promising directions along which the work described in this chapter could have continued. However, two very different problems, one pragmatic and one intellectual, stood in our way. The pragmatic issue was the need for large address space and personal lisp machines; the intellectual issue was our increased awareness that we did not really know what it it meant to "understand" how a complex piece of equipment works. In particular we did not know what mental models the experts had of a given system's functioning, nor did we know how these models were learned, for they certainly weren't explicitly taught.

The computational issue that we faced with SOPHIE III was that it barely fits into the address space of a PDP-10 (256K). SPICE, SOPHIE I and SOPHIE III each already occupied its own separate address space so nothing could be gained by running SOPHIE III stand-alone. This made it extremely difficult to extend SOPHIE III, especially to expand its coaching capabilites along the lines developing out of our coaching research on mathematical games. Address space limitations were not our only concern: To do "formative" evaluations, discovering the *reasons* for the pedagogical success and failures of our system, we needed the speed of efficient, dedicated Lisp machines. Without this speed, student reactions to the long and unpredictable delays often swamped our experimental probes.

5.1 MENTAL MODELS AND EXPLANATIONS

The issue concerning the need for a theory of human understanding of complex systems, in particular circuits, was clearly the more challenging one. Indeed, much of our recent research has been directed at attacking this problem. It quickly became clear to us that the work that went into SOPHIE II and III on explanation put the cart before the horse. We had no adequate theory of what it meant to understand a circuit and hence no well defined "target" model of what we wanted the student to learn. As a consequence no real theory of explanation was forthcoming.

In our experiments with SOPHIE I and II we substantiated that the beginner and expert alike prefer to reason about the circuit in qualitative and causal terms. The students preferred qualitative explanations and were only comfortable about their understanding if it was in terms of a qualitative causal mechanism. SOPHIE III's only reference to causality is in the precompiled explanations for its circuit-specific rules. It could not generate new ones nor could it expand old ones. A significant step toward the necessary robust theory of causality was achieved the following year by de Kleer, in his doctoral dissertation presenting a theory of qualitative causal reasoning about electronics [de Kleer 1979]. Although his target system is primarily concerned with building robust systems, his theory has turned out to provide a basis for constructing human-oriented explanations.

Our fledgling theory of mental models [de Kleer & Brown 80] is based on mapping the system's physical structure into a network of causal relationships among

its parts. The links of this network represent the set of ways in which one part can affect another, and define a limited causal epistemology in the spirit of Reiger [1977]. The causal network is a mental model that is, metaphorically speaking, "runnable in the mind's eye." It is more than just a qualitative simulation or envisionment for it makes explicit causal attributions to the events unfolding in the simulation. But it is derivable from the above mentioned envisionments.

It is important to realize that these mental models are not equivalent to various animations of how a system's behavior unfolds. At best, such animations *implicitly* manifest the underlying causality of a system and the observer must attribute the right kind of causal link to each observable internal event. A theory of mental models makes explicit this attribution process.

Since these mental models are at least in principle executable they are subject to the same problems as are the implict assumptions buried in simulations of a given system. This casts a new light on all of the robustness considerations that we discussed in the SOPHIE I section since the ideal mental model that we wish to eventually impart to the user should be as free as possible from hidden assumptions. Attempting to characterize ideal mental models, or at least ones with maximal probability of being useful for answering unanticipated questions or predicting the consequences of novel casualties, has led us to explore a set of esthetic principles for critiquing the models — even those which are consistent with the known facts. These principles also provide a gradient or direction for the progress of learning. The goal of this learning is to maximize the robustness of the models

Our central esthetic is the "no function in structure" principle. This principle states that the rules for specifying the behavior of any constitutent part of the overall system should in no way refer, even implicitly, to how the overall system functions. Fully satisfying this principle ensures that the behaviors of each of the system's parts can be represented and understood in a context-free manner, independent of the overall system. Failing to adhere to this principle leads to the understanding of how the system functions being predicated on some aspect of how it functions (i.e., a circularity).

Since this principle suggests, a direction along which the learner can progress, it not only establishes a target mental model, but suggests possible *sequences* of explanations. Explanations themselves, however, should not necessarily obey these principles. Often it is best to initially violate the "no function in structure" principle, thereby providing an easily understand explanation from which to develop a more robust mental models. A good overall explanation would be a sequence of models in which the amount of function in structure decreases.

Perhaps the most unexpected aspect of this project has been how our ideas for building efficient and robust reasoning engines have provided useful distinctions and techniques for attacking the purely cognitive issues of mental models. This fact is even more remarkable in the case of SOPHIE I since no attempt was made to make its reasoning schemes psychologically relevant. An underlying theme for mental models has turned out to be the manipulation of underlying assumptions, just as it did in our reasoning and troubleshooting schemes in SOPHIE III. We see this as the unifying mechanism for any future SOPHIE-like system: The utility of assumptions is that they are a central vehicle for qualitative reasoning.

Acknowledgments

The SOPHIE project has been influenced and helped by numerous people. We are especially indebted to Alan Bell for the substantial amount of work he did on SOPHIE I. Richard Rubenstein and Ned Benhaim helped a great deal on SOPHIE II. We are also grateful to Ed Gardner for making possible the original SOPHIE project and to Harry O'Neil and Dexter Fletcher for their constant encouragement and willingness to shelter the project from various bureaucratic constraints. The project was initially funded by the Air Force Human Resource Laboratory at Lowry Air Force Base and later by DARPA/CTO and the Tri-Service training laboratories. Numerous people have read preliminary drafts and have made invaluable suggestions. Bruce Buchanan, Bill Clancey, Jaime Carbonell, Dan Bobrow and Rachel Rutherford deserve special thanks for their patience and insights.

References

BOBROW, R. J. (1978). *The RUS System.* Bolt Beranek and Newman, Inc., Report 3878.

BROWN, A. L. (1976). *Qualitative Knowledge, Causal Reasoning, and the Localization of Failures.* MIT AI TR-362.

BROWN, J. S. (1977). Remarks on building expert systems. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pp. 1003–1005.

BROWN, J. S. & BURTON, R. (1975). Multiple representations of knowledge for tutorial reasoning. In *Representation and Understanding* (D. Bobrow and A. Collins, eds). Academic Press, New York.

BROWN, J. S. & BURTON, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, **2**, 155–198.

BROWN, J. S., BURTON, R. R. & BELL, A. G. (1974). *SOPHIE. A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An example of AI in CAI).* Bolt Beranek and Newman, Inc., Report 2790.

BROWN, J. S., BURTON, R. R. & BELL, A. G. (1975). SOPHIE. A step towards a reactive learning environment. *International Journal of Man Machine Studies*, **7**, 675–696.

BROWN, J. S., RUBINSTEIN, R. & BURTON, R. R. (1976). *Reactive Learning Environment for Computer Assisted Electronics Instruction.* Bolt Beranek and Newman, Inc., Report 3314.

BURTON, R. R. (1976). *Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems.* Bolt Beranek and Newman, Inc., Report 3453, ICAI Report 3.

BURTON, R. R. (1982). Diagnosing a simple procedural skill. Chapter 8 in this volume.

BURTON, R. R. & BROWN, J. S. (1979). Toward a natural-language capability for computer-assisted instruction. In *Procedures For Instructional Systems Development* (H. O'Neil ed.). Academic Press, New York.

BURTON, R. R. & BROWN, J. S. (1982). An investigation of computer coaching for informal learning activities. Chapter 4 in this volume.

BURTON, R. R. & WOODS, W. A. (1976). A compiling system for augmented transition networks. *Proc. Coling*, **76**.

DE KLEER, J. (1976). *Local Methods for Localizing Faults in Electronic Circuits.* MIT AI AIM-394.

DE KLEER, J. (1979). The origin and resolution of ambiguities in causal arguments. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pp. 197–203.

DE KLEER, J. & BROWN, J. S. (1980). Mental models of physical mechanisms and their acquisition. In *Cognitive Skills and their Acquisition.* Erlbaum, New York.

DOYLE, J. (1979). A truth maintenance system. *Artificial Intelligence*, **12**, 231–272.

GROSZ, B. (1978). Discourse knowledge. In *Understanding Spoken Language* (Donald E. Walker, ed.), pp. 229–344. North-Holland, New York.

HENDRIX, G. G. (1977). *The LIFER Manual: A Guide to Building Practical Natural Language Interfaces.* SRI International, Technical Note 138, Menlo Park, California.

LINDSAY, R. K., BUCHANAN, B. G., FEIGENBAUM, E. A. & LEDERBERG, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project.* McGraw-Hill, New York.

NAGEL, L. W. & PEDERSON, D. O. (1973). Simulation program with integrated circuit emphasis. *Proceedings of the Sixteenth Midwest Symposium Circuit Theory.* Waterloo, Canada.

RIEGER, C. & GRINBERG, M. (1977). The declarative representation and procedural simulation of causality in physical mechanisms. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence,* pp. 250–255.

SLEEMAN, D. H. & HENDLEY, R. J. (1982). ACE: a system which analyses complex explanations. Chapter 5 in this volume.

STALLMAN, R. M. & SUSSMAN, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence,* **9**, 135–196.

WEBBER, B. L. (1979). *A Formal Approach to Discourse Anaphora.* Garland Press, New York. (Also published in 1978 by Bolt Beranek and Newman, Inc., TR-3761.)

WOODS, W. A., KAPLAN, R. M. & WEBBER, B. L. (1972). *The Lunar Sciences Natural Language Information System: Final Report.* Bolt Beranek and Newman, Inc., Report 2378.