10

Toward a Natural-Language Capability for Computer-Assisted Instruction[1]

RICHARD R. BURTON and
JOHN SEELY BROWN

THE NATURE AND REQUIREMENTS OF A NATURAL-LANGUAGE SYSTEM

This is a period of dramatic advances in computer technology that should change the way computers are employed in instruction. Technological advances will decrease the cost of computer hardware to the extent that each student will have available computational resources that are currently restricted to a few elite users. Traditional computer-assisted instruction (CAI) paradigms were developed under the assumption that computational power is a scarce resource, and these paradigms are, for the most part, incapable of exploiting the latest technological advances. To use the increased computational power effectively requires a reevaluation of the role of the computer in instructional paradigms, and, in turn, a reevaluation of the authoring aids needed to facilitate efficient development in this medium.

[1] This research was supported in part by the Defense Advanced Research Projects Agency, Air Force Human Resources Laboratory, Army Research Institute for Behavioral and Social Sciences, and Naval Personnel Research and Development Center under contract number MDA903-76-C-0108. Views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of these agencies or of the United States government.

ISBN 0-12-526660-X

The type of instructional system that we see emerging has specific knowledge and problem-solving expertise that is used to aid students. First, as a source of information, it can answer their questions, evaluate their theories, and critique their solution paths. Second, as a tutorial mechanism, it can form models of both the students' states of knowledge and their reasoning strategies. These structural models are used both to identify fundamental misconceptions and to determine when and how to provide remediation, heuristic recommendations ("hints"), or further instruction.

In general, we are not focusing on techniques for teaching factual, textbook knowledge. Computer-assisted instruction systems that do not use the knowledge they contain (as a textbook does not use the knowledge it contains) can competently handle this task and are inherently cheaper for it. Instead, we are focusing on techniques for teaching *procedural knowledge* and *reasoning strategies* that are learned when students must use their factual knowledge in hands-on laboratory or problem-solving tasks. While the students are getting a chance to exercise their knowledge, the "intelligent" instructional systems that we are considering here attempt to mimic the capabilities of a laboratory instructor. The system works on a one-to-one basis with students, carefully diagnosing what they know, how they reason, and what kinds of deficiencies exist in their ability to apply factual knowledge. The system then uses this inferred knowledge of the students together with its knowledge of pedagogy to determine how best to advance their learning.

Although we are still a long way from attaining this goal, we have developed an organization for intelligent instructional systems (described in Brown, 1977) that appears fruitful. Our methodology for developing this organization (and the theory underlying it) has been to explore parts of the overall organization in *paradigmatic systems*. A paradigmatic system is an easily modified prototype system constructed over a carefully chosen domain of knowledge. This methodology allows experimentation with some aspect of the overall system by simplifying other aspects. We have developed systems for such domains as electronic troubleshooting—SOPHIE (Brown, Burton, & Bell, 1975; Brown, Rubinstein, & Burton, 1976); arithmetic drill and practice—WEST (Burton & Brown, 1976, 1978); elementary algebra (Brown, Burton, & Bell, 1975); and procedural skills in arithmetic—BUGGY (Brown & Burton, 1978). In addition, systems of similar spirit are being developed by Carr and Goldstein (1977).

One of the major stumbling blocks for an intelligent instructional system is the lack of a natural means of communication between the student and the computer. This chapter addresses the problems of using natural language (English) as the communication language for advanced

computer-based instructional systems. The instructional environment places requirements on a natural-language understanding system that exceed the capabilities of all existing systems. These requirements include (*a*) efficiency; (*b*) habitability; (*c*) tutorial capability; and (*d*) the ability to exist with ambiguity. However, there are major leverage points within the instructional environment that allow these requirements to be met. In the remainder of this section, we will elaborate on these requirements.

A primary requirement for a natural-language processor, in an instructional situation, is *efficiency*. Imagine the following setting: The student is at a terminal actively working on a problem. The student decides that another piece of information is needed to advance the solution, so a query is formulated. Having finished typing the question, the student will wait for the system to give an answer before continuing to work on the solution. During the time it takes the system to understand the query and generate an answer, the student is apt to forget pertinent information and lose interest. Psychological experiments have shown that response delays longer than 2 seconds have serious effects on the performance of complex tasks via terminals (Miller, 1968). In these 2 seconds the system must understand the query; deduce, infer, look up, or calculate the answer; and generate a response. Another adverse effect of poor response time is that more of the student's searching for the answer is done internally (i.e., without using the system). This decreases the amount of information the tutoring system receives and increases the amount of induction that must be performed, making the problem of figuring out what the student is doing much harder (e.g., students will not "show their work" when solving a problem; they will just present the answer).

The second requirement for a natural-language processor is *habitability*. Any natural-language system written in the foreseeable future is not going to be able to understand all of natural language. What a good natural-language interface must do is characterize and understand a usable subset of the language. Watt (1968) defines a "habitable" sublanguage as "one in which its users can express themselves without straying over the language boundaries into unallowed sentences [p. 338]." Very intuitively, for a system to be habitable it must, among other things, allow the user to make local or minor modifications to an accepted sentence and get another accepted sentence. Exactly how much modification constitutes a minor change has never been specified. Some examples may provide more insight into this notion.

1. *Is anything wrong?*
2. *Is there anything wrong?*

3. *Is there something wrong?*
4. *Is there anything wrong with Section 3?*
5. *Does it look to you as if Section 3 could have a problem?*

If a natural-language processor accepts Sentence 1, it should also accept the modifications given in Sentences 2 and 3. Sentence 4 presents a minor syntactic extension that may have major repercussions in the semantics but that should also be accepted. Sentence 5 is an example of a possible paraphrase of Sentence 4 that is beyond the intended notion of habitability. Based on the acceptance of Sentences 1–4, the user has no reason to expect that Sentence 5 will be handled.

Any sub-language that does not maintain a high degree of habitability is apt to be worse than no natural-language capability at all because, in addition to the problem one is seeking information about, the student is faced, sporadically, with the problem of getting the system to understand a query. This second problem can be disastrous both because it occurs seemingly at random and because it is ill-defined.

In an informal experiment to test the habitability of a system, the authors asked a group of four students to write down as many ways as possible of asking a particular question. The original idea was to determine how many of the various paraphrasings would be accepted by the prototype systems we were testing. The students each came up with one phrasing very quickly but had tremendous difficulty thinking of any others, even though three of the first phrasings were different! This experience demonstrates the lack of the student's ability to do "linguistic" problem solving and points out the importance of accepting the student's first phrasing.

An equally important aspect of the habitability problem is multisentence (or dialogue) phenomena. When students use a system that exhibits "intelligence" through its inference capabilities, they quickly start to assume that the system must also be intelligent in its conversational abilities as well. For example, they will frequently delete parts of their statements that they feel are obvious, given the context of the preceding statements. Often they are totally unaware of such deletions and show surprise and/or anger when the system fails to utilize contextual information as clearly as they (subconsciously) do. The use of context manifests itself in the use of such linguistic phenomena as pronominalizations, anaphoric deletions, and ellipses. The following sequence of questions exemplifies these problems:

6. *What is the population of Los Angeles?*
7. *What is it for San Francisco?*
8. *What about San Diego?*

The third requirement for a natural-language processor is that it be *self-tutoring* (i.e., that it should teach the students about its capabilities). As the students use the system, they should begin to feel the range and limitations of the sub-language. When the students use a sentence that the system cannot understand, they should receive feedback that will enable them to determine why it cannot. There are at least two kinds of feedback. The simplest (and most often seen) merely provides some indication of what parts of the sentence caused the problem (e.g., unknown word or phrase). A more useful kind of feedback goes on to provide a response based on those parts of the sentence that did make sense and then indicate (or give examples of) possibly related, acceptable sentences. It may even be advantageous to have the system recognize common unacceptable sentences and in response to them, explain why they are not in the sub-language. (See the fifth section, on experiences with SOPHIE, for further discussion of this point.)

The fourth requirement for a natural-language system is that it be aware of *ambiguity*. Natural language gains a good deal of flexibility and power by not forcing every meaning into a different surface structure. This means that the program that interprets natural language sentences must be aware that more than one interpretation is possible. For example, when asked

9. *Was John believed to have been shot by Fred?*

one of the most potentially disastrous responses is "Yes." The user may not be sure whether Fred did the shooting or the believing or both. More likely, the user, being unaware of any ambiguity, assumes an interpretation that may be different than the system's. If the system's interpretation is different, the user thinks he has received the answer to his query when in fact he has received the answer to a completely independent query. Either of the following is a much better response:

10. *Yes, it is believed that Fred shot John.*
11. *Yes, Fred believes that John was shot.*

The system need not necessarily have tremendous disambiguation skills, but it must be aware that misinterpretations are possible and inform the user of its interpretation. In those cases where the system makes a mistake the results may be annoying but should not be catastrophic.

This chapter presents the development of a technique that we have named *semantic grammars* for building natural-language processors that satisfy the above requirements. The next section presents a dialogue from the "intelligent" CAI system SOPHIE that we used to refine and demonstrate this technique. This dialogue provides concrete examples of the

kinds of linguistic capabilities that can be achieved using semantic grammars. The third section describes semantic grammar as it first evolved in SOPHIE, and points out how it allows semantic information to be used to handle dialogue constructs and to allow the directed ignoring of words in the input. The fourth section discusses the limitations that were encountered in the evolution of semantic grammars in SOPHIE as the range of sentences was increased and how these might be overcome by using a different formalism—augmented transition networks (ATN). This section also reports on the conversion of the SOPHIE semantic grammar to an ATN, and the extensions to the ATN formalism that were necessary to maintain the solutions presented in the previous section. It also includes comparison timings between the two versions of the natural-language processor. The fifth section describes experiences we have had with SOPHIE and presents techniques developed to handle problems in the area of nonunderstood sentences. The sixth section suggests directions for future work.

DIALOGUE FROM SOPHIE

Before delving into the structural aspects and technical details of the semantic grammar technique, we would first like to provide a concrete example of the dialogues it has supported. This section presents an annotated dialogue of a student using the "Intelligent" CAI system SOPHIE.[2] SOPHIE was developed to explore the use of artificial intelligence techniques in providing tutorial feedback to students engaged in problem-solving activities. The particular problem-solving activity that SOPHIE is concerned with is the troubleshooting of a malfunctioning piece of electronic equipment. SOPHIE models the piece of equipment and answers the student's requests for measurements and other information to aid in debugging the equipment. More important, throughout the problem-solving session, SOPHIE can evaluate the logical consistency of a student's hypothesis or generate hypotheses that are consistent with the behavior the student has thus far observed.[3] In the dialogue, the student's typing is underlined. Even though the dialogue deals with electronic jargon, the linguistic issues it exemplifies occur in all domains. The

[2] The dialogue is intended to demonstrate SOPHIE's linguistic capabilities and, though it touches upon the major features of SOPHIE, it is not meant to exhibit the logical or deductive capabilities of the system.

[3] The reader is encouraged to see Brown and Burton (1975) for further examples and descriptions of SOPHIE's tutorial and inferential capabilities.

annotations (below the dialogue) attempt to point out these problems and should be understandable to the non-electronics-oriented reader.

```
WELCOME TO SOPHIE—A SIMULATED ELECTRONICS LABORATORY.
```
The circuit (Figure 10.1) is based on the Heathkit IP-28 power supply. The IP-28 is a reasonably sophisticated power supply with both current-limiting and voltage-limiting behavior. These two interrelated feedback loops make troubleshooting this circuit nontrivial.

```
≫INSERT A FAULT
```
The Student tells SOPHIE to make one of the components in the circuit bad. SOPHIE randomly selects a fault, inserts it into a model of the instrument, and tells the student how the front panel controls are presently set.

```
THE INSTRUMENT HAS A FAULT AND ITS PRESENT CONTROL SETTINGS ARE:
CC      1.0     CURRENT CONTROL—FROM .0 (MIN CURRENT) TO 1.0
CR      HIGH    CURRENT RANGE SWITCH, HIGH=1 AMP, LOW=.1 AMP
LOAD    1000    LOAD RESISTANCE IN OHMS
VC      1.0     VOLTAGE CONTROL—FROM .0 (MIN VOLTAGE) TO 1.0
VR      HIGH    VOLTAGE RANGE SWITCH, HIGH=30, LOW=10 VOLTS
≫WHAT IS THE OUTPT?
OUTPT ⇒ OUTPUT
THE OUTPUT VOLTAGE IS 1.2 VOLTS.
```
SOPHIE corrects the student's typing mistake. Notice that the output was disambiguated as the output voltage.

```
IF YOU ARE NOT SURE WHETHER A VALUE IS CORRECT, YOU CAN ASK "IS THAT VALUE
CORRECT?"
```
The system points out a feature that the student may not otherwise be aware of. Correct here means what it is in a working instrument.

```
≫IS THAT RIGHT?
IN A WORKING CIRCUIT THE OUTPUT VOLTAGE IS 30.1 VOLTS.
THAT VALUE IS INCORRECT.
```
SOPHIE permits reference to previous objects in the dialogue. Here, That refers to the value of the previous measurement.

```
≫WHAT IS THE VOLTAGE BETWEEN THE CURRENT LIMITING TRANSISTOR
. . . AND THE CONSTANT CURRENT SOURCE?
THE VOLTAGE AT N4 IS 2.48 VOLTS.
```
This is an example of some of the complexities that arise in even limited domains. Although the syntax of the sentence is voltage between x and y, it really means voltage at z with respect to ground where z is the point between x and y. The output routines rephrase the input in a simpler form (i.e., between x and y reduces to N4) so that the user knows the system's interpretation and has an easier way of referring to that location.

```
≫WHAT IS IT BETWEEN THERE AND THE BASE OF Q6?
THE VOLTAGE BETWEEN N4 AND N5 IS 1.28 VOLTS.
```
Here it refers to the voltage and there refers to N4. Also notice that

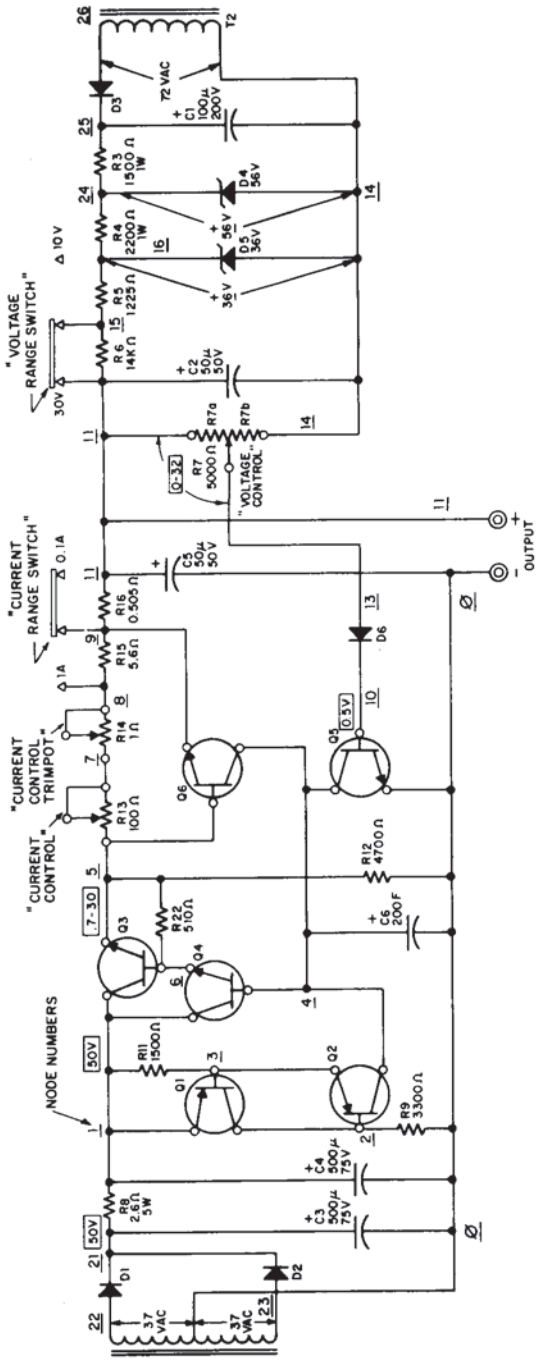

Figure 10.1 Circuit schematic of the power supply.

between has to be treated differently—that is, two nodes are explicit now.

≫IS THERE ANY RIPPLE IN THE VOLTAGE AT THE BASE OF Q4?
I'M SORRY, I DO NOT HANDLE AC CHARACTERISTICS.

SOPHIE recognizes that the student is moving into an area that it is unable to handle. This is currently done by keyword recognition (e.g., ripple) but points out a technique that should be incorporated in other aspects of the system as well.

≫WHAT IS THE BASE CURRENT OF Q3?
THE BASE CURRENT OF Q3 IS 12 MICROAMPS.

The following collection of sentences demonstrates the conciseness and flexibility of the dialogue mechanism. After introducing a context in a complete sentence, the student can easily and naturally ask related questions.

≫THRU THE EMITTER?
THE EMITTER CURRENT OF Q3 IS −.26 MILLIAMPS.

This type of sentence fragment, which we will refer to as an ellipsis, occurs frequently in dialogues. Rather than specifying a complete thought, an ellipsis gives only that piece of the thought that contrasts an earlier thought. Notice that the referent cannot be determined on syntactic grounds (as is possible with the ellipsis "the emitter current"); yet this is the sort of statement a student might make if the last answer had caused a slight shifting of his thinking.

≫Q4
THE EMITTER CURRENT OF Q4 IS −.26 MILLIAMPS.

In this case, the student is changing his attention to a different transistor. There is ambiguity as to whether he wanted the base current, the emitter current, or both. SOPHIE picks the most recent one and gives its interpretation as a complete sentence.

≫IF C2 WERE LEAKY, WHAT WOULD THE VOLTAGE ACROSS IT BE?

As this sentence shows, the referent to it can occur earlier in the same sentence.

IS IT OKAY IF I USE 10000 FOR THE LEAKAGE RESISTANCE OF C2?
> YES
IF THE LEAKAGE RESISTANCE OF C2 WERE 10000
UNDER THE PRESENT SETTINGS, THE VOLTAGE ACROSS C2 WOULD BE 26.3 VOLTS.
IN A WORKING CIRCUIT IT IS 28.9 VOLTS.

≫IF Q6 HAS AN OPEN EMITTER AND A SHORTED BAE COLLECTOR JUNCTION WHAT HAPPENS TO THE VOLTAGE BETWEEN ITS BASE AND THE JUNCTION OF THE VOLTGE LIMITING SECTION AND THE VOLTAGE REFERENCE SOURCE?
BAE ⇒ BASE
VOLTGE ⇒ VOLTAGE

Several examples of the use of conjunction. Imagine the student's frustration if forced to type this query over because of spelling mistakes.

UNDER THE SETTINGS:
CC=9, CR=LOW, LOAD=10, VC=1.0, VR=LOW.
THE VOLTAGE BETWEEN N5 AND N13 WOULD BE 8.94 VOLTS.
IN A WORKING CIRCUIT IT IS 9.57 VOLTS.

```
≫WHAT IS THE CURRENT THRU THE CC WHEN THE VC IS SET TO 1.0?
WITH THE VOLTAGE CONTROL NOW SET AT 1.0
THE CURRENT THRU THE CURRENT CONTROL IS 30 MILLIAMPS.
    CC and VC are abbreviations. The controls are also "parts" that can be
    used in measurements.
≫WHAT IS IT WITH IT SET TO .8?
WITH THE VOLTAGE CONTROL NOW SET AT .8
THE CURRENT THRU THE CURRENT CONTROL IS 24 MILLIAMPS.
    Here the system must correctly instantiate two occurrences of it. The
    referent to the second it could be either CC or VC, since these are both
    controls.
```

SEMANTIC GRAMMAR

Introduction

In the first section we described the requirements for a natural-language processor in a learning environment. Briefly, they are efficiency and friendliness over the class of sentences that arise in a dialogue situation. The major leverage points we have that allow us to satisfy these requirements are (*a*) limited domain; (*b*) limited activities within that domain; and (*c*) known conceptualizations of the domain. In other words, we know the problem area, the type of problem the students are trying to solve, and the way they should be thinking about the problem in order to solve it. What we are then faced with is taking advantage of these constraints in order to provide an effective communication channel.

Notice that all of these constraints relate to concepts underlying the student's activities. In SOPHIE, the concepts include voltage, current, parts, transistors, terminals, faults, particular parts, hypotheses, controls, settings of controls, and so on. The dependency relationships between concepts include things such as these: Voltage can be measured at terminals, parts can be faulted, and controls can be set. The student, in formulating a query or statement, is requesting information or stating a belief about one of these relationships (e.g., *What is the voltage at the collector of transistor Q5?* or *I think resistor R9 is open.)*

It occurred to us that the best way to characterize the statements used for this task was in terms of the concepts themselves as opposed to the traditional syntactic structures. The language can be described by a set of grammar rules that characterize, for each concept or relationship, all of the ways of expressing it in terms of other constituent concepts. For example, the concept of a measurement requires a quantity to be measured and something against which to measure it. A measurement is

typically expressed by giving the quantity followed by a preposition, followed by the thing that specifies where to measure (e.g., *voltage across capacitor C2, current thru diode D1).* These phrasings are captured in this grammar rule (this is not actually a rule from the grammar but is merely intended to be suggestive):

<MEASUREMENT> := <MEASURABLE/QUANTITY> <PREP> <PART>

The concept of a measurement can, in turn, be used as part of other concepts—for example, to request a measurement (*What is the voltage across capacitor C2?*) or to check a measurement (*Is the current thru divide D1 correct?*) We call this type of grammar a *semantic grammar* because the relationships it tries to characterize are semantic and conceptual as well as syntactic.

Semantic grammars have two advantages over traditional syntactic grammars. They allow semantic constraints to be used to make predictions during the parsing process, and they provide a useful characterization of those sentences that the system should try to handle. The predictive aspect is important for four reasons:

1. It reduces the number of alternatives that must be checked at a given time.
2. It reduces the amount of syntactic (grammatical) ambiguity.
3. It allows recognition of ellipsed or deleted phrases.
4. It permits the parser to skip words at controlled places in the input (i.e., it enables a reasonable specification of control).

These points will be discussed in detail in a later section.

The characterization aspect is important for two reasons:

1. It provides a handle on the problem of constructing a habitable sublanguage. The system knows how to deal with a particular set of tasks over a particular set of objects. The sublanguage can be partitioned by tasks to accept all straightforward ways of expressing those tasks, but does not need to worry about others.
2. It allows a reduction in the number of sentences that must be accepted by the language while still maintaining habitability. There may be syntactic constructs that are used frequently with one concept (task) but seldom with another. For example, relative clauses may be useful in explaining the reasons for performing an experimental test but are an awkward (though possible) way of requesting a measurement. By separating the processing along semantic grounds, one may gain efficiency by not having to accept the awkward phrasing.

Representation of Meaning

Since natural-language communication is the transmission of concepts via phrases, the "meaning" of a phrase is its correspondent in the conceptual space. The entities in SOPHIE's conceptual space are objects, relationships between objects, and procedures for dealing with objects. The meaning of a phrase can be a simple data object (e.g., *current limiting transistor*) or a complex data object (e.g., *C5 open; Voltage at Node 1*). The meaning of a question is a call to a procedure that knows how to determine the answer. The meaning of a command is a call to a procedure that performs the specified action. (Declarative statements are treated as requests because the pragmatics of the situation imply that the student is asking for verification of his statement. For example, *I think C2 is shorted* is taken to be a request to have the hypothesis *C2 is shorted* critiqued.) For example, the procedural specialist DOFAULT knows how to fault the circuit and is used to represent the meaning of commands to fault the circuit (e.g., *Open R9; Suppose C2 shorts and R9 opens*). The argument that DOFAULT needs in order to perform its task is an instance of the concept of faults that specifies the particular changes to be made (e.g., *R9 being open*). These same concepts of particular faults also serve as arguments to two other specialists: HYPTEST, which determines the consistency of a fault with respect to the present context (e.g., *Could R9 be open*) and SEEFAULT, which checks the actual status of the circuit (e.g., *Is R9 open?*).

Result of the Parsing

Basing the grammar on conceptual entities allows the semantic interpretation (the determination of the concept underlying a phrase) to proceed in parallel with the parsing. Since each of the nonterminal categories in the grammar is based on a semantic unit, each grammar rule can specify the semantic description of a phrase that it recognizes in much the same way that a syntactic grammar specifies a syntactic description. The construction portion of the rules is procedural. Each rule has the freedom to decide how the semantic descriptions, returned by the constituent items of that rule, are to be put together to form the correct "meaning."

For example, the meaning of the phrase *Q5* is the data base object Q5. The meaning of the phrase *the collector of Q5* is (COLLECTOR Q-5), where COLLECTOR is a function that returns the data base item that is the collector of the given transistor.

The rule for <MEASUREMENT> expresses all of the ways that the

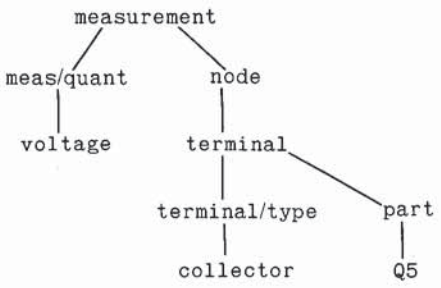

Figure 10.2 Control structure for the <measurement> rule.

student can give a measurable quantity and also supply its required arguments. The structure that results from <MEASUREMENT> is a function call to the function MEASURE that supplies the quantity being measured and other arguments specifying where to measure it. Thus the meaning of the phrase *the voltage at the collector of Q5* is (MEASURE VOLTAGE (COLLECTOR Q5)), which was generated from the control structure (see Figure 10.2).

The grammar rule for <MEASUREMENT> also accepts "meaningless" phrases such as *the power dissipation of Node 4.* In addition, it accepts some meaningful phrases such as *the resistance between Node 3 and Node 14,* which SOPHIE does not calculate. This results from generalizing together concepts that are not treated identically in the surface structure. In this case, *voltage, current, resistance,* and *power dissipation* were generalized to the concept of a measurable quantity. The advantage of allowing the grammar to accept more statements and having the argument checking done by the procedural specialists is that the semantic routines provide the feedback as to why a sentence cannot be interpreted or "understood." It also keeps the grammar from being cluttered with special rules for blocking meaningless phrases. Carried to the limit, the generalization strategy would return the grammar to being "syntactic" again (e.g., all data objects are "noun phrases"). The trick is to leave semantics in the grammar when it is beneficial—to stop extraneous parsings early, or to tighten the range of a referent for an ellipsis or deletion. This is obviously a task-specific trade-off. (Bobrow and Brown [1975] describe an interesting paradigm from which to consider this trade-off.)

The relationship between a phrase and its meaning is usually straightforward. However, it is not limited to simple embedding. Consider the phrases *the base emitter of Q5 shorted* and *the base of Q5 shorted to the emitter*. The thing which is "shorted" in both of these phrases is the "base emitter junction of Q5." The rule that recognizes both of these

phrases, <PART/FAULT/SPEC>, can handle the first phrase by invoking its constituent concepts of <JUNCTION> (*base emitter of Q5*) and <FAULT/TYPE> (*shorted*) and combine their results. In the second phrase, however, it must construct the proper junction from the separate occurrences of the two terminals involved.

This discussion has been presented as if the concepts were defined a priori by the capabilities of the system. Actually, for the system to remain at all habitable, the concepts are discovered in the interplay between expanding the corpus of sentences the system can handle and adding capabilities to the system. When a particular English construct is difficult to handle, it is probably an indication that the concept it is trying to express has not been recognized properly by the system. In our example *the base of Q5 is shorted to the emitter,* the relationship between the phrase and its meaning is awkward because the present concept of shorting requires a part or a junction. The example is getting at a concept of shorting, in which any two terminals can be shorted together (e.g., *the positive terminal of R9 is shorted to the anode of D6*). This is a viable conceptual view of shorting, but its implementation requires allowing arbitrary changes in the topology of the circuit, which is beyond the efficiency limitations of SOPHIE's simulator. Thus, the system we were working with led us to define the concept in too limited a way.

Use of Semantic Information during Parsing

Prediction

Having described the notion of a semantic grammar, we will now describe the ways in which it allows semantic information to be used in the understanding process. One use of semantic grammars is to predict the possible alternatives that must be checked at a given point. Consider, for example, the phrase *the voltage at xxx*. After the word *at* is reached in the top-down, left-to-right parse, the grammar rule corresponding to the concept *measurement* can predict very specifically the conceptual nature of *xxx*: It must be a phrase that directly or indirectly specifies a location in the circuit. For example, *xxx* could be *the junctions of the current limiting section and the voltage reference source* but cannot be *3 ohms*.

Semantic grammars also have the effect of reducing the amount of grammatical ambiguity. In the phrase *the voltage at xxx,* the prepositional phrase *at xxx* will be associated with the noun *voltage* without considering any alternative parses that associate it somewhere higher in the tree.

Predictive information is also used to aid in the determination of referents for pronouns. If the above phrase were *the voltage at it,* the grammar would be able to restrict the class of possible referents to locations.

By taking advantage of the available sentence contexts to predict the semantic class of possible referents, the referent-determination process is greatly simplified. For example:

1a. *Set the voltage control to .8?*
1b. *What is the current thru R9?*
1c. *What is it with it set to .9?*

In 1c, the grammar is able to recognize that the first *it* refers to a measurement that the student would like retaken under slightly different conditions. The grammar can also decide that the second *it* refers to either a potentiometer or to the load resistance (i.e., one of those things that can be set). The referent for the first *it* is the measurement taken in 1b, *the current thru R9*. The referent for the second *it* is *the voltage control*, which is an instance of a potentiometer. The context mechanism that selects the referents will be discussed later.

Simple Deletion

The semantic grammar is also used to recognize simple deletions. The grammar rule for each conceptual entity knows the nature of that entity's constituent concepts. When a rule cannot find a constituent concept, it can either

a. fail (if the missing concept is considered to be obligatory in the surface structure representation) or,
b. hypothesize that a deletion has occurred and continue.

For example, the concept of a TERMINAL has as one of its realizations the constituent concepts of a TERMINAL-TYPE and a PART. When its grammar rule finds only the phrase *the collector*, it uses this information to posit that a part has been deleted (i.e., TERMINAL-TYPE gets instantiated to *the collector* but nothing gets instantiated to PART). The natural-language processor then uses the dependencies between the constituent concepts to determine that the deleted PART must be a TRANSISTOR. The "meaning" of this phrase is then *the collector of some transistor*. **Which** transistor is determined when the meaning is evaluated in the present dialogue context. In particular, the semantic form returned is the function PREF and the classes of possible referents; in our example the form would be (COLLECTOR (PREF (TRANSISTOR))).

Ellipsis

Another use of the semantic grammar allows the processor to recognize elliptic utterances. These are utterances that do not express complete thoughts—a completely specified question or command—but only give

differences between the intended thought and an earlier one.[4] For example, 2b, 2c, and 2d are elliptic utterances.

2a. *What is the voltage at Node 5?*
2b. *At Node 1?*
2c. *And Node 2?*
2d. *What about between nodes 7 and 8?*

Ellipses can begin with introductory phrases such as *and* in 2c or *what about* in 2d; however this is not required, as can be seen in 2b. Part of the ellipsis rule is given in Figure 10.3.

```
<ELLIPSIS> := [<ELLIPSIS/INTRODUCER>] <REQUEST/PIECE> !
              [<ELLIPSIS/INTRODUCER>] if <PART/FAULT/SPEC>
<REQUEST/PIECE> := [<PREP>] <NODE> !
                   [<PREP>] <PART> !
                   between <NODE> and <NODE> !
                   [<PREP>] <JUNCTION> !
                   etc.
```

Figure 10.3 Ellipsis rule.

The grammar rule identifies which concept or class of concepts is possible from the context available in the elliptic utterance.

Though the parser is usually able to determine the intended concepts from the context available in an elliptic utterance, this is not always the case. Consider the following two sequences of statements.

3a. *What is the voltage at Node 5?*
3b. *10?*

4a. *What is the output voltage if the load is 100?*
4b. *10?*

In 3b, *10* refers to Node 10, whereas in 4b it refers to a load of 10. The problem this presents to the parser is that the concepts underlying these two elliptic utterances have nothing in common except their surface realizations. The parser, which operates from conceptual entities, does not have a concept that includes both of these interpretations. One solution would be to have the parser find all parses (concepts) and then choose between them on the basis of context. Unfortunately, this would mean that time is wasted looking for more than one parse for the large percentage of sentences in which it is not necessary to do so. A better solution

[4] The standard use of the word *ellipsis* refers to any deletion. Rather than invent a new word, we shall use the restricted meaning here.

would be to allow structure among the concepts, so that the parser would recognize *10* as a member of the concept *number*. Then the routines that find the referent would know that numbers can be either node numbers or values. This type of recognition could profitably be performed by a bottom-up approach to parsing. However, its advantages over the present scheme are not enough to justify the expense incurred by a bottom-up parse to find all possible well-formed constituents. At present, the parser assumes one interpretation, and a message is printed to the student indicating the assumed interpretation. If it is wrong, the student must supply more context in his request. In fact, *10?* is taken as a load specification and if the student meant the node he would have to use *at 10*, *N10*, or *Node 10*. Later we will discuss the mechanism that determines to which complete thought an ellipsis refers.

Using Context to Determine Referents

Pronouns and Deletions

Once the parser has determined the existence and class (or set of classes) of a pronoun or deleted object, the context mechanism is invoked to determine the proper referent. This mechanism has a history of student interactions during the current session that contains, for each interaction, the parse (meaning) of the student's statement and the response calculated by the system. This list provides the range of possible referents and is searched in reverse order to find an object of the proper semantic class (or one of the proper classes). To aid in the search, the context mechanism knows how each of the procedural specialists appearing in a parse uses its arguments. For example, the specialist MEASURE has a first argument that must be a quantity and a second argument that must be a part, a junction, a section, a terminal, or a node. Thus when the context mechanism is looking for a referent that can be either a PART or a JUNCTION, it will look at the second argument of a call to MEASURE but not the first. Using the information about the specialists, the context mechanism looks in the present parse and then in the next most recent parse, etc., until an object from one of the specified classes is found.

The significance of using the specialist to filter the search instead of just keeping a list of previously mentioned objects is that it avoids misinterpretations due to object–concept ambiguity. As an example, consider the following sequence from the sample dialogue in the previous section:

5. *What is the current thru the CC when the VC is 1.0?*
6. *What is it when it is .8?*

Sentence 5 will be recognized by the following rules from the semantic grammar:

```
$1) <REQUEST> := <SIMPLE/REQUEST> when <SETTING/CHANGE>
$2) <SIMPLE/REQUEST> := what is <MEASUREMENT>
$3) <MEASUREMENT> := <MEAS/QUANT> <PREP> <PART>
$4) <SETTING/CHANGE> := <CONTROL> is <CONTROL/VALUE>
$5) <CONTROL> := VC
```

with a resulting semantic form of:

```
(RESETCONTROL (STQ VC 1.0)
              (MEASURE CURRENT CC))
```

RESETCONTROL is a function whose first argument specifies a change to one of the controls and whose second argument consists of a form to be evaluated in the resulting instrument context. STQ is used to change the setting of one of the controls. The first argument to MEASURE gives the quantity to be measured. The second specifies where it is to be measured. To recognize Sentence 6, the application of Rules $2 and $5 are changed. There is an alternative rule for <SIMPLE/REQUEST> that looks for those anaphora (i.e., *that, it,* and *one*) that refer to a measurement. These phrases, such as *it, that result,* or *the value,* are recognized by the nonterminal <MEASUREMENT/PRONOUN>. The alternative to $2 that would be used to parse (6) is

```
<SIMPLE/REQUEST> := what is <MEASUREMENT/PRONOUN>.
```

The semantics of <MEASUREMENT/PRONOUN> indicate that an entire measurement has been deleted. The alternative to Rule $5,

```
<CONTROL> := it,
```

recognizes *it* as an acceptable way to specify a control. The resulting semantic form for Sentence 6 is

```
(RESETCONTROL (STQ (PREF '(CONTROL)) .8)
              (PREF '(MEASUREMENT)))
```

The function PREF searches back through the context of previous semantic forms to find the most recent mention of a member of one of the classes. In the above example, it will find the control VC but not CC because the character imposed on the arguments of MEASURE is that of a "part," not a "control." The presently recognized classes for deletions are PART, TRANSISTOR, FAULT, CONTROL, POT, SWITCH, DIODE, MEASUREMENT, and QUANTITY. (The members of the classes are derived from the semantic network associated with a circuit.)

Referents for Ellipses

If the problem of pronoun resolution is looked upon as finding a previously mentioned object for a currently specified use, then the problem of ellipsis can be thought of as finding a previously mentioned use for a currently specified object. For example,

7. *What is the base current of Q4?*
8. *In Q5?*

The given object is *Q5,* and the earlier function is *base current.* For a given elliptic phrase, the semantic grammar identifies the concept (or class of concepts) involved. In 7, since Q5 is recognized by the nonterminal <TRANSISTOR/SPEC>, the class would be TRANSISTOR. The context mechanism then searches for a specialist in a previous parse that accepted the given class as an argument. When one is found, the new phrase is placed in the proper argument position and the modified parse is used as the meaning of the ellipsis.

Limitations to the Context Mechanism

The method of semantic classification (to determine reference) is very efficient and works well over our domain. It definitely does not solve all the problems of reference. Charniak (1972) has pointed out the substantial problems of reference in a domain as seemingly simple as children's stories. One of his examples demonstrates how much world knowledge may be required to determine a referent: "Janet and Penny went to the store to get presents for Jack. Janet said 'I will get Jack a top.' 'Don't get Jack a top,' said Penny. 'He has a top. He will make you take *it* back [p. 7]'."

Charniak argues that to understand to which of the two tops "it" refers requires knowing about presents, stores and what they will take back, etc. Even in domains where it may be possible to capture all of the necessary knowledge, classification may still lead to ambiguities. For example, consider the following:

9. *What is the voltage at Node 5 if the load is 100?*
10. *Node 6?*
11. *7?*

In Statement 11 the user means Node 7. In Statement 10, he has reinforced the use of ellipsis as referring to node number. (For example, when Statement 10 is left out, Statement 11 is much more awkward.) On the other hand, if Statement 11 had been *1000* or if Statement 10 had been *10?,*

things would be more problematic. When Statement 11 is *1000,* we can infer that the user means a load of 1000 because there is no Node 1000. If Statement 10 had been *10?,* there would be genuine ambiguity slightly favoring the interpretation as a load because that was the last number mentioned. The major limitation of the current technique, which must be overcome in order to tackle significantly more complicated domains, is its inability to return more than one possible referent. It considers each one individually until it finds one that is satisfactory. The amount of work involved in employing a technique that allows comparing referents has not been justified by our experience.

Fuzziness

Having the grammar centered around semantic categories allows the parser to be sloppy about the actual words it finds in the statement. Having a concept in mind, and being willing to ignore words to find it, is the essence of keyword parsing schemes. It is effective in those cases where the words that have been skipped either are redundant or specify gradations of an idea that are not distinguished by the system. For example, in the sentence, *Insert a very hard fault, very* would be ignored; this is effective because the system does not have any further structure over the class of hard faults. In the sentence, *What is the voltage across resistor R8? resistor* can be ignored because it is implied by *R8.* (The first of these examples could be handled by making *very* a noise word (i.e., deleting it from all sentences). *Resistor,* however, is not a noise word in all cases (e.g., *What is the current through the current sensing resistor?*) and hence cannot be deleted.

One advantage that a procedural encoding of the grammar (discussed later) has over pattern-matching schemes in the implementation of fuzziness is its ability to control exactly where words can be ignored. This provides the ability to blend pattern-matching parsing of those concepts that are amenable to it with the structural parsing required by more complex concepts. The amount of fuzziness—how many, if any, words in a row can be ignored—is controlled in two ways. First, whenever a grammar rule is invoked, the calling rule has the option of limiting the number of words that can be skipped. Second, each rule can decide which of its constituent pieces or words are required and how tightly controlled the search for them should be. In SOPHIE, the normal mode of operation of the parser is tight in the beginning of a sentence, but fuzzier after it has made sense out of something.

Fuzziness has two other advantages worth mentioning briefly. It reduces the size of the dictionary because all known noise words do not

have to be included. In those cases where the skipped words are meaningful, the misunderstanding may provide some clues to the user that allow him to restate his query.

Preprocessing

Before a statement is parsed, a preprocessor performs three operations. The first expands abbreviations, deletes known noise words, and canonicalizes similar words to a common form. The second is a cursory spelling correction. The third is a reduction of compound words.

Spelling correction is attempted on any word of the input string that the system does not recognize. The spelling correction algorithm[5] takes the possibly misspelled word and a list of correctly spelled words and determines which, if any, of the correct words is close to the misspelled word (using a metric determined by number of transpositions, doubled letters, dropped letters, etc.). During the initial preprocessing, the list of correct words is very small (approximately a dozen) and is limited to very commonly misspelled words and/or words that are critical to the understanding of a sentence. The list is kept small so that the time spent attempting spelling correction, prior to attempting a parse, is kept to a minimum. Remember that the parser has the ability to ignore words in the input string, so we do not want to spend a lot of time correcting a word that will not be needed in understanding the statement. But notice that certain words can be critical to the correct understanding of a statement. For example, suppose that the phrase *the base emitter current of Q3* were incorrectly typed as *the bse emitter current of Q3*. If *bse* were not recognized as being *base*, the parser would ignore it and misunderstand the phrase as *the emitter current of Q3*, a perfectly acceptable but much different concept.[6] Because of this problem, words like *base*, which if ignored have been found to lead to misunderstandings, are considered critical, and their spelling is corrected before any parse is attempted. Other words that are misspelled are not corrected until the second attempt at spelling correction that is done after a statement fails to parse.

Compound words are single concepts that appear in the surface structure as a fixed series of more than one word. Their reduction is very important to the efficient operation of the parser. For example, in the

[5] The spelling correction routines are provided by INTERLISP and were developed by Teitelman for use in the DWIM facility (Teitelman, 1969, 1974).

[6] To minimize the consequences of such misinterpretation, the system always responds with an answer that indicates what question it is answering, rather than just giving the numeric answer.

question, *What is the voltage range switch setting?, voltage range switch* is rewritten as the single item *VR*. If not rewritten, *voltage* would be mistaken as the beginning of a measurement (as in *What is the voltage at N4?*) and an attempt would have to be made to parse *range switch setting* as a place to measure voltage. Of course, after this failed, the correct parse could still be found, but reducing compound words helps to avoid search. In addition, the reduction of compound words simplifies the grammar rules by allowing them to work with larger conceptual units. In this sense, the preprocessing can be viewed as a preliminary bottom-up parse that recognizes local, multiword concepts.

Implementation

Once the dependencies between semantic concepts have been expressed in the Backus–Naur Form (BNF), each rule in the grammar is encoded (by hand) as a procedure in the programming language LISP. This encoding process imparts to the grammar a top-down control structure, specifies the order of application of the various alternatives of each rule, and defines the process of pattern matching each rule. The resulting collection of LISP functions constitutes a goal-oriented parser in a fashion similar to SHRDLU (Winograd, 1973), but without the backtracking ability of PROGRAMMER.

As has been argued elsewhere (Winograd, 1973; Woods, 1970), encoding the grammars as procedures—including the notion of process in the grammar—has advantages over using traditional phrase structure grammar representations. Four of these advantages are

1. The ability to collapse common parts of a grammar rule while still maintaining the perspicuity of the grammar
2. The ability to collapse similar rules by passing arguments (as with SENDR)
3. The ease of interfacing other types of knowledge (in SOPHIE, primarily the semantic network) into the parsing process
4. The ability to build and save arbitrary structures during the parsing process. (This ability is sometimes provided by allowing augments on phrase structure rules.)

In addition to the advantages it shares with other procedural representations, the LISP encoding has the computational advantage of being compilable directly into efficient machine code. The LISP implementation is efficient because the notion of process it contains (one process doing recursive descent) is close to that supported by physical machines, whereas those of ATN and PROGRAMMER are nondeterministic and

hence not directly translatable into present architecture. (See Burton [1976] for a description of how it is possible to minimize this mismatch.)

In terms of efficiency, the LISP implementation of the semantic grammar succeeds admirably. The grammar written in the INTERLISP dialect of LISP (Teitelman, 1974) can be block-compiled. Using this technique, the complete parser takes about 5K of storage and parses a typical student statement consisting of 8 to 12 words in around 150 milliseconds!

A NEW FORMALISM—SEMANTIC AUGMENTED TRANSITION NETWORKS

Using the techniques described in the previous section, a natural-language processor capable of supporting the dialogue presented in the second section and requiring less than 200 milliseconds cpu time per question was constructed. In addition, these same techniques were used to build a processor for NLS-SCHOLAR (Grignetti, Gould, Hausmann, Bell, Harris, & Passafiume, 1974; Grignetti, Hausmann, & Gould, 1975) (built by K. Larkin), and an interface to an experimental laboratory for exploring mathematics using attribute blocks (Brown & Burton, 1978). In the construction of these varying systems, the notion of semantic grammar proved to be useful. The LISP implementation, however, was found to be a bit unwieldy. Although expressing the grammar as programs is efficient and allows complete freedom to explore new extensions, the technique is lacking in perspicuity. This lack of perspicuity has three major drawbacks: (*a*) the difficulty encountered when trying to modify or extend the grammar; (*b*) the problem of trying to communicate the extent of the grammar to either a user or a colleague; (*c*) the problem of trying to reimplement the grammar on a machine that does not support LISP. These difficulties have been partially overcome by using a second, parallel representation of the grammar in a specification language similar to the Backus–Naur Form, which is the representation we have been presenting throughout this report. This, however, requires supporting two different representations of the same information and does not really solve problems *a* or *c*. The solution to this problem is a better formalism for expressing and thinking about semantic grammars. This section discusses such a formalism.

Augmented Transition Networks (ATN)

Some years ago, Chomsky (1957) introduced the notion that the processes of language generation and language recognition could be viewed

in terms of a machine. One of the simplest of such models is the finite state machine. It starts off in its initial state looking at the first symbol, or word, of its input sentence and then moves from state to state as it gobbles up the remaining input symbols. The sentence is **accepted** if the machine stops in one of its final states after having processed the entire input string; otherwise the sentence is **rejected**. A convenient way of representing a finite state machine is as a transition graph, in which the states correspond to the nodes of the graph and the transitions between states correspond to its arcs. Each arc is labeled with a symbol whose appearance in the input can cause the given transition.

In an augmented transition network, the notion of a transition graph has been modified in three ways: (*a*) the addition of a recursion mechanism that allows the labels on the arcs to be nonterminal symbols that correspond to networks; (*b*) the addition of arbitrary conditions on the arcs that must be satisfied in order for an arc to be followed; (*c*) the inclusion of a set of structure-building actions on the arcs, together with a set of named registers for holding partially built structures. (This discussion follows closely a similar discussion in Woods [1970], to which the reader is referred. A reader familiar with the augmented transition network formalism may wish to skip to the section "Advantages to the Augmented Transition Network Formalism.") Figure 10.4 is a specification of a language for representing augmented transition networks. The specification is given in the form of an extended, context-free grammar in which alternative ways of forming a constituent are represented on separate lines and the symbol + is used to indicate arbitrarily repeatable constituents. (+ is used to mean 0 or more occurrences. Though the accepted usage of + is 1 or more, the accepted symbol for 0 or more, *, has not been used to avoid confusion with the use of the symbol * in the augmented transition network formalism.) The nonterminal symbols are lowercase English descriptions enclosed in angle brackets. All other symbols, except +, are terminals. Nonterminals not given in Figure 10.4 have names intended to be self-explanatory.

The first element of each arc is a word indicating the type of arc. For arcs of type CAT, WRD, and PUSH, the arc type together with the second element corresponds to the label on an arc of a state transition graph. The third element is an additional test. A CAT (category) arc can be followed if the current input symbol is a member of the lexical category named on the arc and if the test on the arc is satisfied. A PUSH (network call) arc causes a recursive invocation of a lower level network beginning at the state indicated, if the test is satisfied. The WRD (word) arc can be followed if the current input symbol is the word named on the arc and if the test is satisfied. The TST (test) arc can be followed if the test is

```
<transition network> := (<arc set><arc set>+)
<arc set> := (<state> <arc>+)
<arc> := (CAT <category name> <test> <action>+ <term act>)
         (WRD <word> <test> <action>+ <term act>)
         (PUSH <state> <test> <action>+ <term act>)
         (TST <arbitrary label> <test> <action>+ <term act>)
         (POP <form> <test>)
         (VIR <constituent name> <test> <action>+ <term act>)
         (JUMP <state><test><action>+)
<action> := (SETR <register> <form>)
            (SENDR < register><form >)
            (LIFTR <register> <form>)
            (HOLD <constituent name> <form>)
            (SETF <feature> <form>)
<term act> := (TO <state>)
<form> :=(GETR <register>)
          LEX
          *
          (GETF <form> <feature>)
          (BUILDQ <fragment> <register>+)
          (LIST <form>+)
          (APPEND <form> <form>)
          (QUOTE <arbitrary structure>)
```

Figure 10.4 A language for representing ATNs.

satisfied (the label is ignored). The VIR arc (virtual arc) can be followed if a constituent of the named type has been placed on the hold list by a previous HOLD action and the constituent satisfies the test. In all of these arcs, the actions are structure-building actions, and the terminal action specifies the state to which control is passed as a result of the transition. After CAT, WRD, and TST arcs, the input is advanced; after VIR and PUSH arcs it is not. The JUMP arc can be followed whenever its test is satisfied, control being passed to the state specified in the second element of the arc without advancing the input. The POP (return from network) arc indicates the conditions under which the state is to be considered a final state and the form of the constituent to be returned.

The actions, forms, and tests on an arc may be arbitrary functions of the register contents. Figure 10.4 presents a useful set that illustrates major features of the ATN. The first three actions specified in Figure 10.4 cause the contents of the indicated register to be set to the value of the indicated form. SETR (set register) causes this to be done at the current level of computation, SENDR (send register) at the next lower level of embedding, so that information can be sent down during a PUSH, and LIFTR (lift register) at the next higher level of computation, so that additional information can be returned to higher levels. The HOLD action places a

form on the HOLD list to be used at a later place in the computation by a VIR arc. SETF (set feature) provides a means of setting a feature of the constituent being built.

GETR (get register value) is a function whose value is the contents of the named register. LEX (lexical item) is a form whose value is the current input symbol. The asterisk (*) is a form whose value depends on the context of its use:

1. In the actions of a CAT arc, the value of * is the root form of the current input word.
2. In the actions of a PUSH arc, it is the value of the lower computation.
3. In the actions following a VIR arc, the value of it is the constituent removed from the HOLD list.

GETF is a function that determines the value of a specified feature of the indicated form (which is usually *). BUILDQ is a general structure-building form that places the values of the given registers into a specified tree fragment. Specifically, it replaces each occurrence of + in the tree fragment with the contents of one of the registers (the first register replacing the first occurrence of +, the second register the second, etc.). In addition, BUILDQ replaces occurrences of * by the value of the form *. The remaining three forms make a list out of the specified arguments (LIST), append two lists together to make a single list (APPEND), and produce as a value the (unevaluated) arbitrary form (QUOTE).

Advantages of ATN Formalism

The augmented transition network (ATN) formalism was seriously considered at the beginning of the SOPHIE project but rejected as being too slow. In the course of developing the LISP grammar, it became clear that the primary reason for a significant difference in speed between an ATN grammar and a LISP grammar is due to the fact that processing the augmented transition network (ATN) is an interpreted process, whereas LISP is compilable and therefore the time problem could be overcome by building an ATN compiler. During the period of evolution of SOPHIE's grammar, an ATN compiler was constructed (see Burton, 1976). In the next section we will discuss the advantages we hoped to gain by using the ATN formalism.

These advantages fall into three general areas: (*a*) conciseness; (*b*) conceptual effectiveness; and (*c*) available facilities. By conciseness we mean that writing a grammar as an ATN takes fewer characters than LISP. The ATN formalism gains conciseness by not requiring the speci-

fication of details in the parsing process at the same level required in LISP. Most of these differences stem from the fact that the ATN assumes it has a machine whose operations are designed for parsing, whereas LISP assumes it has a lambda calculus machine. For example, a lambda calculus machine assumes a function has one value. A function call to look for an occurrence at a nonterminal while parsing (in ATN formalism, a PUSH) must return at least two values: the structure of the constituent found, and the place in the input where the parsing stopped. A good deal of complexity is added to the LISP rules in order to maintain the free variable that has to be introduced to return the structure of the constituent. Other examples of unnecessary details include the binding of local variables and the specification of control structure as ANDs and ORs.

The conciseness of the ATN results in a grammar that is easier to change, easier to write and debug, and easier to understand, and hence provide for better communication. We realize that conciseness does not necessarily lead to these results (APL being a counter example in computer languages, mathematics in general being another); however, this is not a problem. The correspondence between the grammar rules in LISP and ATN is very close. The concepts that were expressed as LISP code can be expressed in nearly the same way as ATN but in fewer symbols.

The second area of improvement deals with conceptual effectiveness. Loosely defined, conceptual effectiveness is the degree to which a language encourages one to think about problems in the right way. One example of conceptual effectiveness can be seen by considering the implementation of case-structured rules. (See Bruce [1975] for a discussion of case systems.) In a typical case-structure rule, the verb expresses the function (or relation name) and the subject, and the object and prepositional phrases express the arguments of the function or relation. Let us assume for the purpose of this discussion that we are looking at four different cases (agent, location, means, and time) of the verb GO—*John went to the store by car at 10 o'clock*. In a phrase structure rule-oriented formalism one would be encouraged to write:

```
<statement> :=<actor> <action/verb> <location> <means> <time>
```

Since the last three cases can appear in any order, one must also write five other rules:

```
<statement> :=<actor> <action/verb> <location> <time> <means>
   ⋮
```

In an ATN one is inclined toward a graph (see Figure 10.5) that expresses more clearly the case structure of the rule. There is no reason why in the

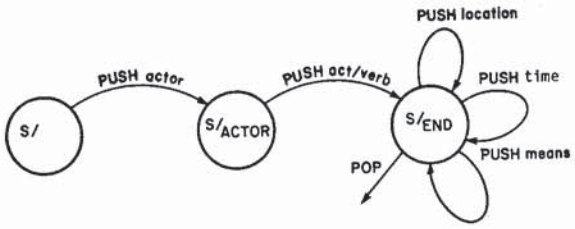

Figure 10.5 A case structure ATN rule.

LISP version of the grammar one could not write loops that are exactly analogous to the ATN (the ATN compiler, after all, produces such code!). However, a rule-oriented formalism does not encourage one to think this way. An alternative rule implementation is

```
<action>:=<actor><action/verb><actionl>
<actionl>:=<actionl><time>
<actionl>:= <actionl><location>
<actionl>:= <actionl><means>
```

This is easier (shorter) to write but it has the disadvantage of being left-recursive. To implement it, one is forced to write the LISP equivalent of the augmented transition network that creates a difference between the rule representation and the actual implementation. This method also has the disadvantage of introducing the nonterminal <actionl> into the grammar.

Another conceptual advantage of the ATN framework is that it encourages the postponing of decisions about a sentence until a differential point is reached, thereby allowing potentially different paths to stay together. In the rule-oriented SOPHIE grammar there are top-level rules for <set>, a command to change one of the control settings, and <modify>, a command to fault the instrument in some way. Sentence 1 is a <set> and Sentence 2 is a <modify>.

1. *Suppose the current control is high.*
2. *Suppose the current control is shorted.*

The two parse paths for these sentences should be the same for the first five words, but they are separated immediately by the rules <set> and <modify>. An ATN encourages structuring the grammar so that the decision between <set> and <modify> is postponed so that the paths remain together. It could be argued that the fact that this example occurred in SOPHIE's grammar is a complaint against top-down parsing or semantic grammars, or just our particular instantiation of a semantic grammar. We suspect the latter but argue that rule representations encourage this type of behavior.

Another conceptual aid provided by ATNs is their method of handling ambiguity. Our LISP implementation uses a recursive descent technique (which can alternatively be viewed as allowing only one process). This requires that any decision between two choices be made correctly because there is no way to try out the other choice **after** the decision is made. At choice points, a rule can, of course, "look ahead" and gain information on which to base the decision, similar to the "wait-and-see" strategy used by Marcus (1975), but there is no way to back up and remake a decision once it has returned.

The effects of this can be most easily seen by considering the lexical aspects of the parsing. A prepass collapses compound words, expands abbreviations, etc. This allows the grammar to be much simpler because it can look for units like *voltage–control* instead of having to decode the noun phrase *voltage control.* Unfortunately, without the ability to handle ambiguity, this rewriting can be done only on words that have no other possible meaning. So, for example, when the grammar is extended to handle

3. *Does the voltage control the current limiting section?*

the compound *voltage–control* would have to be removed from the prepass rules and included in the grammar. This reduces the amount of bottom-up processing that can be done and results in a slower parse. It also makes compound rules difficult to write because all possible uses of the individual words must be considered to avoid errors. Another example is the use of the letter "C" as an abbreviation. Depending on context, it could possibly mean either current, collector, or capacitor. Without allowing ambiguity in the input, it could not be allowed as an abbreviation unless explicitly recognized by the grammar.

The third general area in which ATNs have an advantage is in the available facilities to deal with complex linguistic phenomena. Though our grammar has not yet expanded to the point of requiring any of the facilities, the availability of such facilities cannot be ignored as an argument favoring one approach over another. A primary example is the general mechanism for dealing with coordination in English described in Woods (1973).

Conversion to Semantic ATN

For the reasons discussed above, the SOPHIE semantic grammar was rewritten in the ATN formalism. We wish to stress here that the rewriting was a process of **changing form** only. The content of the grammar remained the same. Since a large part of the knowledge encoded by the

grammar continues to be semantic in nature, we call the resulting grammar a *semantic ATN*.

Figure 10.6 presents the graphic ATN representation of semantic grammar nonterminal, which recognizes the straightforward way of expressing a terminal of a part in the circuit—the base of Q5, the anode of it, the collector. It also shows a simple example of how the recognition of anaphoric deletions can be captured in ATN formalism. By the state TERMINAL/TYPE, both the determiner and the terminal type—base, anode—have been found. The first arc that leaves TERMINAL/TYPE accepts the preposition that begins the specification of the part. The second arc (JUMP arc) corresponds to hypothesizing that the specification of the part has been deleted, as in *The base is open*. The action on the arc builds a place-holding form that identifies the deletion and specifies (from information associated with the terminal type that was found) the classes of objects that can fill the deletion. The method for determining the referent of the deletion remains the same as described in the third section.

The SOPHIE semantic ATN is compiled using the general ATN compiling system described in Burton (1976). The SOPHIE grammar provides the compiling system with a good contrast to the LUNAR grammar (Woods, Kaplan & Nash-Webber, 1972) (that was used as a test during development of the compiler), since it does not use many of the potential features. In addition, a bench mark, of sorts, was available from the LISP implementation of the grammar that could be used to determine the computational cost of using the ATN formalism.

There were two modifications made to the compiling system to improve its efficiency for the SOPHIE application. In the SOPHIE grammar, a large number of the arcs check for the occurrence of particular words. When there is more than one arc leaving a state, the ATN formalism requires that all of these arcs be tried, even if more than one of these is a WRD (word) arc and an earlier WRD arc has succeeded. This is especially costly, since the taking of an arc requires the creation of a configuration (data structure) to try the remaining arcs. In those cases when the grammar writer knows that none of the other arcs can succeed, this should be avoided. As a solution to this problem, the GROUP arc type was added. The GROUP arc allows a set of contiguous arcs to be designated as

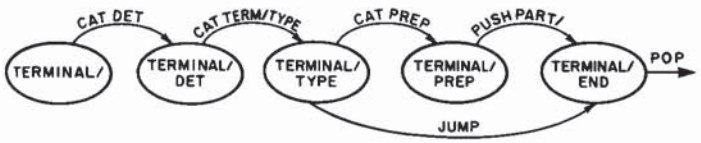

Figure 10.6 A semantic ATN that recognizes deletion.

mutually exclusive. The form of the GROUP arc is (GROUP arc1 arc2 . . . arcn). The arcs are tried, one at a time, until the conditions on one of the arcs are met. This arc is then taken, and the remaining arcs in the GROUP are forgotten—not tried. If a PUSH arc is included in the GROUP, it will be taken if its test is true, and the remaining arcs will not be tried even if the PUSHed-for constituent is not found. For example, consider the following grammar state:

```
(S/1
  (GROUP (CAT A T (TO S/2))
         (WRD X T (TO S/3))
         (CAT B T (TO S/4))))
```

At most, one of the three arcs will be followed. Without GROUPing them together, it is possible that all three might be followed—if the word *X* had interpretations as both Category A and Category B.

The GROUP arc also provides an efficient means of encoding optional constituents. The normal method of allowing options in ATN is to provide an arc that accepts the optional constituent and a second arc that jumps to the next state without accepting anything. For example, if in State S/2 the word *very* is optional, the following two arcs would be created:

```
(S/2
  (WRD VERY T (TO REST-OF-S/2))
     (JUMP REST-OF-S/2 T))
```

The inefficiency arises when the word *very* does occur. The first arc is taken, but an alternative configuration that will try the second arc must be created, and possibly later explored. When these arcs are embedded in a GROUP, the alternative will not be created, thus saving time and space. As a result, it will not have to be explored, possibly saving more time. A warning should be included here that the GROUP arc can reject sentences that might otherwise be accepted. In our example, *very* may be needed to get out of the state REST-OF-S/2. In this respect, the GROUP arc is a departure from the original ATN philosophy that arcs should be independent. However, for some applications, the increased efficiency can be critical.

The other change to the compiling system (for the semantic grammar application) dealt with the preprocessing operations. The preprocessing facilities described in the last section included (*a*) lexical analysis to extract word endings; (*b*) a substitution mechanism to expand abbreviations, delete noise words, and canonicalize synonyms; (*c*) dictionary retrieval routines; and (*d*) a compound word mechanism to collapse multiword phrases. For the SOPHIE application we added the ability to use the INTERLISP spelling correction routines and the ability to derive word definitions from SOPHIE's semantic net. The extraction of def-

initions from the semantic network for part names and node names reduces the size of the dictionary and simplifies the operation of changing circuits. In addition, a mechanism called MULTIPLES was developed that permits string substitution within the input. This is similar to the notion of compounding, but differs in that a compound rule creates an alternative lexical item, whereas the multiple rule creates a different lexical item. After the application of a compound rule, there is an additional edge in the input chart; after a multiple rule, the effect is the same as if the user had typed in a different string.

Fuzziness

The one aspect of the LISP implementation that has not been incorporated into the ATN framework is fuzziness, the ability to ignore words in the input. Although we have not worked out the details, the nondeterminism provided by ATNs lends itself to an interesting approach. In a one-process—recursive descent—implementation, the rule that checks for a word must decide (with information passed down from higher rules) whether to try skipping a word, or give up. The critical information that is not available when this decision has to be made is whether or not there is another parse that would use that word. In the ATN, it is possible to suspend a parse and come back to it after all other paths have been tried. Fuzziness could be implemented so that rather than skip a word and continue, it can skip a word and suspend, waiting for the other parses to fail or suspend. The end effect may well be that sentences are allowed to get fuzzier because there is no danger of missing the correct parse.

Comparison of Results

The original motivation for changing to the ATN was its perspicuity. As Winograd (1973) has pointed out, simple grammars are perspicuous in almost any formalism; complex grammars are still complex in any formalism. We found the ATN formalism much easier to think in, write in, and debug. The examples of redundant processing that were presented earlier in this section were discovered while converting to ATN. For a gross comparison on conciseness, the ATN grammar requires 70% fewer characters to express than the LISP version.

The efficiency results were surprising. Table 10.1 gives comparison timings between the LISP version and the ATN compiled version. As can be seen, the ATN version takes less than twice as much time. This was pleasantly counterintuitive, since we expected the LISP version to be much faster because of the amount of hand optimization that had been

done while encoding the grammar rules. In presenting the comparison timing, it should be mentioned that there are three differences between the two systems that tended to favor the ATN version. (The exact extent to which each of these differences contributed is difficult to gather statistics on because of the INTERLISP block compiler that gains efficiency by hiding internal workings. The exact contribution of each could certainly be determined but was not deemed worth the effort.) One difference is the lack of fuzziness in the ATN version. The LISP version spent time testing words other than the current word, looking ahead to see whether it was possible to skip this word, which was not done in the ATN version. The second is the creation of categories for words during the preprocessing in the ATN version that reduced the amount of time spent accessing the semantic net and hence reduced the time required to perform a category membership test in the ATN system. The third is the simplification of the grammar and increase in the amount of bottom-up processing that could be done because of the ambiguity allowed in the input chart. In our estimation, the lack of fuzziness is the only difference that may have had a significant effect, and this can be included explicitly in the ATN in places where it is critical, by using TST arcs and suspend actions, without a noticeable increase in processing time. In conclusion, we are very pleased with the results of the compiled semntantic ATN and feel that the ATN compiler makes the ATN formalism computationally efficient enough to be used in real systems.

TABLE 10.1
Comparison of ATN versus LISP Implementation

Times (in seconds) are "prepass" + "parsing."
1. *What is the output voltage?*
LISP – .024 + .018 = .042
ATN – .048 + .033 = .081
2. *What is the voltage between there and the base of Q6?*
LISP – .038 + .039 = .077
ATN – .090 + .046 = .136
3. *Q5?*
LISP – .010 + .046 = .056
ATN – .013 + .060 = .073
4. *What is the output voltage when the voltage control is set to .5?*
LISP – .045 + .038 = .083
ATN – .096 + .048 = .144
5. *If Q6 has an open emitter and a shorted base collector junction, what happens to the voltage between its base and the junction of the voltage limiting section and the voltage reference source?*
LISP – .206 + .188 = .394
ATN – .259 + .090 = .349

EXPERIENCES WITH SOPHIE AND TECHNIQUES FOR HANDLING PROBLEMS

When we began developing a natural-language processor for an instructional environment, we knew it had to be (*a*) fast; (*b*) habitable; (*c*) self-tutoring; and (*d*) able to deal with ambiguity. The basic conclusion that has arisen from the work presented here is that it is possible to satisfy these constraints. The notion of semantic grammar presented earlier provides a paradigm for organizing the knowledge required in the understanding process that permits efficient parsing. In addition, semantic grammar aids the habitability by providing insights into a useful class of dialogue constructs, and permits efficient handling of such phenomena as pronominalizations and ellipses. The need for a better formalism for expressing semantic grammars led to the use of augmented transition networks. The ability of the ATN-expressed semantic grammar to satisfy the above stated requirements is demonstrated in the natural language front-end for the SOPHIE system.

A point that needs to be stressed is that the SOPHIE system has been (and is being) used by uninitiated students in experiments to determine the pedagogical effectiveness of its environment. Although much has been learned about the problems of using a natural-language interface, these experiments were not debugging sessions for the natural-language component. The natural-language component has unquestionably reached a state at which it can be conveniently used to facilitate learning about electronics. In this section, we will describe the experiences of students using the natural-language component, and present some ideas on handling erroneous inputs.

Impressions, Experiences, and Observations

As mentioned in the introduction, students are very unskilled at paraphrasing their thoughts. This same inability to perform linguistic paraphrase carried over to the actual interaction with SOPHIE via terminal. Whenever the system did not accept a query, there was a marked delay before the student tried again. Sometimes the student would abandon a line of questioning completely. At the same time, data collected over many sessions indicated that there was no standard—canonical—way to phrase a question. Table 10.2 provides some examples of the range of phrasings used by students to ask for the voltage at a node. As Table 10.2 shows, students are likely to conceive of their questions in many ways and to express each of these conceptions in any of several phrasings. Yet other experiences indicate that they lack the ability to

TABLE 10.2
Sample Student Inputs

The following are some of the input lines typed by students with the intent of discovering the voltage at a node in the circuit.

What is the voltage at node 1?
What is the voltage at the base of Q5?
How much voltage at N10?
And what is the voltage at N1?
N9?
V at the neg side of C6?
V11 is?
What is the voltage from the base of transistor Q5 to ground?
What V at N16?
Coll. of Q5?
Node 16 Voltage?
What is the voltage at Pin 1?
Output?

convert easily to another conceptualization or phrasing. Since the nonacceptance of questions creates a major interruption in the student's thought process, the acceptance of many different paraphrases is critical to maintaining flow in the student's problem solving.

Another interesting phenomenon that occurred during sessions was the change in the linguistic behavior of the students as they used the system. Initially, queries were stated as complete English questions, generally stated in templates created by the students from the written examples of sessions that we had given them. If they needed to ask something that did not exactly fit one of their templates, they would try a minor variant. As they became more familiar with the mode of interaction, they began to use abbreviations, to leave out parts of their questions, and, in general, to assume that the system was following their interaction. After 5 hours of experience with the system, almost all of one student's queries contained abbreviations and one in six depended on the context established by previous statements.

Feedback—When the Grammar Fails

From our experiences with students using SOPHIE, we have been impressed with the importance of providing feedback to unacceptable inputs—doing something constructive when the system does not understand an input. Though it may appear that in a completely habitable system all inputs would be understood, no system has ever attained this goal, and none will in the foreseeable future. To be natural to a naive user, an

intelligent system should also act intelligently when it fails. The first step toward having a system fail intelligently is the identification of possible areas of error. In student's use of the SOPHIE system, we have found the following types of errors to be common:

1. Spelling errors and mistypings—*Shortt the CE og Q3 and opwn its base; What is the vbe Q5?*
2. Inadvertent omissions—*What is the BE of Q5?* (The user left out the quantity to measure. Note that in other domains this is a well-formed question.)
3. Slight misconceptions that are predictable—*What is the output of transistor Q3?* (the output of a transistor is not defined); *What is the current thru Node 1?* (nodes are places where voltage is measured and may have numerous wires associated with them); *What is R9?* (R9 is a resistor); *Is Q5 conducting?* (The laboratory section of SOPHIE gives information that is directly available from a real lab such as currents and voltages.)
4. Gross misconceptions whose underlying meaning is well beyond designed system capabilities—*Make the output voltage 30 volts; Turn on the power supply and tell me how the unit functions; What time is it?*

In the remainder of this section, we will discuss the solutions used in the SOPHIE system to provide feedback.

The use of a spelling correction algorithm (borrowed from INTERLISP) has proven to be a satisfactory solution to typos and misspellings. During one student's session, spelling correction was required on, and resulted in proper understanding of, 10% of the questions. The major failings of the INTERLISP algorithm are the restriction on the size of the target set of correct words (time increases linearly with the number of words) and its failure to correct run-on words. (The time required to determine whether a word may be two—possibly misspelled—words run together increases very quickly with the length of the word and the number of possibly correct words. With no context to restrict the possible list of words, the computation involved is prohibitive.) A potential solution to both shortcomings would be to use the context of the parser to reduce the possibilities when it reaches the unknown word. Because of the nature of the grammar, this would allow semantic context as well as syntactic context to be used.

Of course, the use of any spelling correction procedure has some dangers. A word that is spelled correctly but that the system does not know may be changed through spelling correction to a word the system does know. For example, if the system does not know the word *top* but

does know *stop,* a user's command to *top everything* can be disastrously misunderstood. For this reason, words like *stop* are not spelling-corrected.

Our solution to predictable misconceptions is to recognize them and give error messages that are directed at correcting the misconception. We are currently using two different methods of recognition. One is to loosen up the grammar so that it accepts plausible but meaningless sentences. This technique provides the procedural specialists called by the plausible parse enough context to make relevant comments. For example, the concept of current through a node is accepted by the grammar even though it is meaningless. The specialist that performs measurements must then check its arguments and provide feedback if necessary:

```
⩾ WHAT IS THE CURRENT THRU NODE 4?
The current thru a node is not meaningful since by Kirchoff's law the
sum of the currents thru any node is zero. Currents can be measured
thru parts (e.g., CURRENT THRU C6) or terminals (e.g., CURRENT THRU
THE COLLECTOR OF Q2).
```

Notice that the response to the question presents some examples of how to measure the currents along wires that lead into the mentioned node. Examples of questions that will be accepted and are relevant to the student's needs are among the best possible feedback.

The second method of recognizing common misconceptions is to "key" feedback off single words or groups of words. In the following examples, the keys are *or* and *turned on.* Notice that the response presents a general characterization of the violated limitations as well as suggestions for alternative lines of attack.

```
⩾ COULD Q1 OR Q2 BE SHORTED?
I can handle only 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.

⩾ IS THE CURRENT LIMITING TRANSISTOR TURNED ON?
The laboratory section of SOPHIE is designed to provide the same
elementary measurements that would be available in a real lab. If
you want to determine the state of a transistor, measure the perti-
nent currents and voltages.
```

These methods of coping with errors have proved to be very helpful. However, they require that all of the misconceptions be predicted and programmed for in advance. This limitation makes them inapplicable to novel situations.

The remaining severe problems a user has stem from omissions and major misconceptions. After a simple omission, the user may not see that

he has left anything out and may conclude that the system does not know that concept or phrasing of that concept. For example, when the user types *What is the BE of Q5* instead of *What is the VBE of Q5?*, he may decide that it is unacceptable because the system does not allow *VBE* as an abbreviation of *base emitter voltage*. For conceptual errors, the user may waste a lot of time and energy attempting several rephrasings of his query, none of which can be understood because the system does not know the concept the user is trying to express. For example, no matter how it is phrased, the system will not understand *Make the output voltage 30 volts* because measurements cannot be directly changed; only controls and specifications of parts can be changed.

The feedback necessary to correct both of these classes of errors must identify any concepts in the statement that are understood and suggest the range of things that can be done to–with these concepts. This may help the user see an omission or may suggest alternative conceptualizations that get at the same information (for example, to change the output voltage indirectly by changing one of the controls) or at least provide enough information for the user to decide when to quit.

FUTURE DIRECTIONS

Further Research Areas

The SOPHIE semantic grammar system is designed for a particular context—trouble-shooting—within a particular domain—electronics. It represents the compilation of those pieces of knowledge that are general (linguistic) together with specific domain-dependent knowledge. In its present form, it is unclear which knowledge belongs to which area. The development of semantic grammars for other applications and extensions to the semantic grammar mechanism to include other understood linguistic phenomena will clarify this distinction.

Although the work presented in this chapter has dealt mostly with one area of application, the notion of semantic grammar as a method of integrating knowledge into the parsing process has wider applicability. Two alternative applications of the technique have been completed. One deals with simple sentences in the domain of attribute blocks (Brown & Burton, 1978). Though the sub-language accepted in the attribute-blocks environment is very simple, it is noteworthy that within the semantic grammar paradigm, a simple grammar was quickly developed that greatly improved the flexibility of the input language. The other completed application deals with questions about the editing system NLS (Grignetti *et al.*,

1975). In this application, most questions dealt with editing commands and their arguments, and fit nicely into the case-frame notion mentioned in the fourth section. The case-frame use of semantic grammar is being considered for, and may have its greatest impact on, command languages. Command languages are typically case-centered around the command name that requires additional arguments (its cases). The combination of the semantic classification provided by the semantic grammar and the representation of case rules permitted by ATNs should go a long way toward reducing the rigidity of complex command languages such as those required for message-processing systems. The combination should also be a good representation for natural-language systems in domains where it is possible to develop a strong underlying conceptual space, such as management information systems (Malhotra, 1975).

Conclusions

In the course of this chapter, we have described the evolution of a natural-language processor capable of using complex linguistic knowledge. The guiding strand has been the utilization of semantic information to produce efficient natural-language processors. There were several highlights that represent noteworthy points in the spectrum of useful natural language systems. The procedural encoding technique with fuzziness (third section) allows simple natural-language input to be accepted without introducing the complexity of a new formalism. Encoding the rules as procedures allows flexible control of the fuzziness, and the semantic nature of the rules provides the correct places to take advantage of the flexibility. As the language covered by the system becomes more complex, the additional burden of a grammar formalism will more than pay for itself in terms of ease of development and reduction in complexity. The augmented transition network (ATN) compiling system allows for the consideration of the ATN formalism by reducing its run-time cost, making it comparable to a direct procedural encoding. The natural language front end now used by SOPHIE is constructed by compiling a semantic ATN. As the linguistic complexity of the language accepted by the system increases, the need for more syntactic knowledge in the grammar becomes greater. Unfortunately, this often works at cross-purposes with the semantic character of the grammar. It would be nice to have a general grammar for English syntax that could be used to preprocess sentences; however, one is not forthcoming. A general solution to the problem of incorporating semantics with the current state of incomplete knowledge of syntax remains an open research problem. In the foreseeable future, any system will have to be an engineering trade-off between complexity and

generality on one hand and efficiency and habitability on the other. We have presented several techniques that are viable options in this trade-off.

REFERENCES

Bobrow, R. J., & Brown, J. S. Systematic understanding: Synthesis, analysis, and contingent knowledge in specialized understanding systems. In D. Bobrow & A. Collins (Eds.), *Representation and understanding: Studies in cognitive science*. New York: Academic Press, 1975.

Brown, J. S. Uses of artificial intelligence and advanced computer technology in education. In *Computers and communications*. New York: Academic Press, 1977.

Brown, J. S., & Burton, R. R. Multiple representations of knowledge for tutorial reasoning. In D. Bobrow & A. Collins (Eds.), *Representation and understanding: Studies in cognitive science*. New York: Academic Press, 1975.

Brown, J. S., & Burton, R. R. A paradigmatic example of an artificially intelligent instructional system. *International Journal of Man Machine Studies*, 1978, *10*, 323–340.

Brown, J. S., Burton, R. R., & Bell, A. G. SOPHIE: A step towards a reactive learning environment. *International Journal of Man Machine Studies*, 1975, *7*, 675–696.

Brown, J. S., Rubinstein, R., & Burton, R. R. *Reactive learning environment for computer assisted electronics instruction* (BBN Report No. 3314). Bolt Beranek and Newman Inc., Cambridge, Mass., October 1976.

Bruce, B. C. Case systems for natural language. *Artificial Intelligence*, December 1975, *5*, 327–360.

Burton, R. R. *Semantic grammar: An engineering technique for constructing natural language understanding systems* (BBN Report No. 3453, ICAI Report No. 3). Cambridge, Mass.: Bolt Beranek and Newman Inc., December 1976.

Burton, R. R., & Brown, J. S. A tutoring and student modelling paradigm for gaming environments. *Proceedings for the Symposium on Computer Science and Education*, February 1976,

Burton, R. R., & Brown, J. S. An investigation of computer coaching for informal learning activities. *International Journal of Man Machine Studies*, in press.

Carr, B., & Goldstein, I. *Overlays: A theory of modelling for computer aided instruction* (AI Memo 406). Cambridge: Massachusetts Institute of Technology, February 1977.

Charniak, E. *Toward a model of children's story comprehension* (MIT—TR-266). Cambridge: Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1972.

Chomsky, N. *Syntactic structures*. The Hague: Mouton and Co., 1957.

Grignetti, M. C., Gould, L., Hausmann, C. L., Bell, A. G., Harris, G., & Passafiume, J. *Mixed-initiative tutorial system to aid users of the on-line system (NLS)* (BBN Report No. 2969). Cambridge, Mass.: Bolt Beranek and Newman Inc., November 1974.

Grignetti, M. C., Hausmann, C. L., & Gould, L. An intelligent on-line assistant and tutor—NLS-SCHOLAR. *National Computer Conference*, 1975, *44*, 775–781.

Malhotra, A. *Design criteria for a knowledge-based English language system for management: An experimental analysis*. Unpublished doctoral dissertation, Sloan School of Management, Massachusetts Institute of Technology, 1975.

Marcus, M. Diagnosis as a notion of grammar. In R. Schank & B. L. Nash-Webber (Eds.), *Proceedings of a Workshop on Theoretical Issues in Natural Language Processing*, 1975, *1*, 6–10.

Miller, R. B. Response time in man–computer conversational transactions. In *AFIPS Conference Proceedings* (Fall Joint Computer Conference). Washington: Thompson Book Company, 1968.

Teitelman, W. Towards a programming laboratory. In D. Walker (Ed.), *Proceedings of the International Joint Conference on Artificial Intelligence,* May 1969.

Teitelman, W. *INTERLISP reference manual.* Palo Alto, Calif.: Xerox Palo Alto Research Center, 1974.

Watt, W. C. Habitability. *American documentation,* 1968, *19,* 338–351.

Winograd, T. *Understanding natural language.* New York: Academic Press, 1973.

Woods, W. A. Transition network grammars for natural language analysis. *Communications of the ACM,* 1970, *13,* 591–606.

Woods, W. A. An experimental parsing system for transition network grammars. In R. Rustin (Ed.), *Natural language processing.* New York: Algorithmics Press, 1973.

Woods, W. A., Kaplan, R. M., & Nash-Webber, B. *The lunar sciences natural language information system: final report* (BBN Report #2378). Cambridge, Mass.: Bolt, Beranek and Newman, Inc., 1972.