In W.Kintsch, J.Miller & P.Polson (Eds)
Methods and Tactics in Cognitive
Science. Hillsdale, NJ: Erlbaum.
1984.

9 Competitive Argumentation in Computational Theories of Cognition

Kurt VanLehn
John Seely Brown
Xerox Palo Alto Research Center

James Greeno
University of Pittsburgh

During the past two decades, artificial intelligence and linguistics have had a major impact on the form of theories in cognitive psychology. Prior to about 1960, most theories in cognitive psychology considered information in relatively abstract terms, such as features, items, and chunks. Starting in the 1960s, and increasing during the 1970s, an additional theme in psychological theory has been to take into account the specific information that is present in the tasks that provide the material for theoretical analyses. The difference can be seen, for instance, in psychological analyses of problem solving that were developed in the 1950s, compared with analyses that have been worked out in the 1970s. In the earlier analyses, problem solving was considered as *selection* of a response (solution) that initially had low probability. Factors in the situation were examined for their facilitating or inhibiting effect on selection of the needed response. In computational models, specific task situations are represented, and programs are written that use the information in those representations to simulate processes of actually *constructing* solutions to specific problems.

The capability of analyzing the details of processing specific information is clearly an advance. For example, it enables psychological analyses of human behaviors that one would label "understanding" that are much more detailed than those provided previously. However, there is a well known danger to such an approach. Analyses can become mired in their increased precision and detail, with the result that it is extremely difficult to separate fundamental principles from their supporting detail. Yet explicit principles are needed in order to define or at least constrain the classes of processes and structures that are postulated.

In particular, when such detailed analyses aspire to be empirical theories, they face difficulties in achieving an adequate treatment of individual differences. In most analyses, there has been considerable obscurity in the boundary between what is meant to be true of all subjects, and what is meant to be true of a particular subject. To modify the knowledge base, rules, or other structures in order to fit a model to an individual subject is often easy enough, but how to place well defined limitations on such changes is an open problem. Identifying the universal components of a model and the general principles that constrain the processes is a start toward assessing the "degrees of freedom" of theories that tailor their predictions with non-numeric parameters (such as sets of rules). However, determining the tailorability of such models is at best vaguely understood. Yet it is crucial. A model may have so much tailorability that it can be tuned to match almost any data, rendering it vacuous.

Our concerns are similar to a number of recent contentions that the methodology of artificial intelligence (AI) and related fields is not productive for formulating and defending *theories* of mind (Dresher & Hornstein, 1976; Fodor, 1978/1981; Pylyshyn, 1980, forthcoming; Winograd, 1977). Unlike early criticism of AI (e.g., Dreyfus, 1972), which centered on whether computers could ever be intelligent, these critics concentrate on AI's supposed contributions to psychology. They note that despite its potential, well supported cognitive theories based on AI technology have not been forthcoming. We tend to agree with their conclusion, and offer a brief analysis of why this is so.

Approaches: top down, bottom up, neurological

AI has demonstrated that computer programs can behave with a certain degree of intelligence. However, no consensus has emerged concerning necessary or sufficient design principles for creation of intelligent programs, or even on the limitations imposed by the computational medium on intelligence. It currently appears that the goal of manifesting intelligence *per se* is not in itself constraining enough to force particular architectures or principles to be used.

The failure of AI to demonstrate the existence of "top down" constraints on cognitive theories, principles inherent in the nature of cognition independent of the medium of its implementation, suggests searching "bottom up," starting with concrete instances of intelligent human behavior. That is, given that the goal is to find out how human cognition works, it currently seems advisable to ground the models/analyses in human data. In this chapter, we consider only empirical theories, or rather, the problems of obtaining empirical theories, that maintain the advantages of an AI-like treatment of task information while striving to meet scientific criteria for empirical theories.

The technology of AI adapts readily to a bottom up approach. There are programs that simulate a subject's behavior quite faithfully and in considerable detail (e.g., even eye movements during problem solving can be predicted. Cf., Newell & Simon, 1972). However, despite the addition of empirical responsibility, the current methodology of employing simulation models is still weak in several respects. There has been virtually no attention to the tailorability (degrees of freedom) of simulation models. There has been little argumentation for the individual principles and components of the model. Because the entailments of each principle have not been separated from the performance of the model's simulation as a whole, one is asked to accept the model *in toto* with no explanation as to why it has the principles it does. Although questions of observational adequacy have been treated, questions of descriptive and explanatory adequacy have been almost universally ignored. Indeed, without the additional consideration of tailorability, measures of descriptive and observational adequacy have little meaning.

Some have asserted that it is necessary to add a third kind of constraint by moving to the periphery, where neurological data can be brought to bear on information processing theories. Although this has yielded some exciting results (Marr, 1976; Marr & Nishihara, 1978), the findings obtained thus far seem not to provide strong constraints on the hypotheses about processes such as language comprehension or problem solving.

The missing ingredient for scientific progress is not behavioral or neurological data, we believe, but scientific reasoning that explicates the principles underlying successful models of cognition and connects them with the data. The emphasis must be on the connection; explication alone is not sufficient. Efforts at explicating programs have increased in response to critics (e.g., Kaplan, 1981) who point out that a typical AI/Simulation "explanation" of intelligent behavior is to substitute one black box, a complex computer program, for another, namely the human mind. Extracting the principles behind the design of the computer program is a necessary first step. But many other questions remain to be addressed: What is the relationship between the principles and the behavior? Could the given cognition be simulated if the principles were violated or replaced by somewhat different ones? Would such a change produce inconsistency, or a plausible but as yet unobserved human behavior, or merely a minor perturbation in the predictions? Which alternatives if any can now be rejected in favor of the chosen principles? This connection of explicit principles to the data seem to be critical to progress in computational theories of cognition.

Nature and Importance of Arguments

Computer science has given psychology a new way of expressing models of cognition that is much more detailed and precise than its predecessors. But unfortunately, the increased detail and precision in *stating* models has not been accompanied by correspondingly detailed and precise arguments *analyzing and supporting* them. Consequently, the new, richly detailed models of cognitive science often fail to meet the traditional criteria of scientific theories. By *sup-*

port, we refer to various traditional forms of scientific reasoning such as showing that specified empirical phenomena provide positive or negative evidence regarding hypotheses, showing that an assumption is needed to maintain empirical content and falsifiability, or showing that an assumption has consequences that are contradictory or at least implausible. It is of course not new to desire that cognitive theories have this kind of support. Nor is it a new contribution to point out that current theorizing based on computational models of cognition has been lax in providing such support (Pylyshyn, 1980; Fodor, 1981). Perhaps what is new is discussing how such supporting argumentation could be developed for computational models of the mind.

The focus of this discussion is on the kinds of arguments that are applied to specified theoretical principles. We're not going to advocate building a particular kind of theory, nor do we wish to dispute over criteria for theories (e.g., falsifiability, tailorability). Instead, we discuss what kinds of tools are available or can be fashioned that will help one build computational theories of cognition that will meet some widely accepted standards that have so far proved difficult for such theories to meet. The prime tool of this discussion, actually a class of tools, is the *competitive argument*. Unlike famous technological tools of scientific advance, such as microscopes or Golgi stains, which offered clearer views of tissue structure and other factual material, competitive argument seems to be a tool for analyzing and clarifying the theoretical issues implicit in a computational model of a cognitive faculty.

There often is a great need for such a clarifying instrument when the model under development employs computations. The relationship between a principle and data must often be an indirect one and can take several forms. For instance, a principle might serve as a constraint on a class of processes, perhaps by defining a processing language and an interpreter; the processes in turn might have a mapping to observable behavior defined, for example, by some grain-size assumptions. The indirectness of the relationship between principles and data provides additional reasons for providing clear and adequate supporting argumentation. With appropriate argumentation, it is possible to show, among other things, that it is the principles that are responsible for the computations' empirical coverage and not some obscure or accidental details of the particular computer implementation of the theory. Moreover, the arguments can show that the principles have some force in that they are refutable.

Of course, the idea of argumentation related to specific theoretical principles is not new, and many examples could be listed showing how psychologists have considered principles in relation to their supporting evidence, their testability, and the plausibility of their consequences. However, argumentation regarding specific principles has been relatively rare in computational theories. This is partly because theorists have spent their major effort coming to grips with the precision and subsequent detail of the computational medium and of course with the details of fine-grained, conceptual task analyses.

We view argumentation regarding specific principles as a part of a natural progression. The progression includes stages of task analysis, articulation of principles, and competitive argumentation. In the third section of the chapter, we give an example of the evolution of one particular theory through these stages.

At the beginning of the study of a task domain, a great deal of analysis is necessary before even a crude simulation of behavior is possible. One must learn what details of the task should be included and what can be suppressed. Early examples such as Newell, Shaw, and Simon's (1963) model of proving theorems in logic, Bobrow's (1968) model of solving algebra word problems, and Evans' (1968) model of solving analogy problems (to name just a few examples) were valuable contributions partly because they showed that their mechanisms were *sufficient* for producing correct solutions for an interesting variety of problems in their respective domains. In psychological studies such as Newell and Simon's (1972) models of performance in cryptarithmetic, logic exercises, and chess, and Simon and Kotovsky's (1963) model of solving series completion problems, the sufficiency criterion has been extended to require general similarity to performance by human problem solvers. By and large, the first venture into a task domain does not yield a precise articulation of the principles that structure competence in it. But it is important to emphasize just how difficult it is to forge these first formalizations, and how much is learned from them. Furthermore, the understanding of processes involved in performance provides an essential resource for subsequent investigation of the task domain.

Such a subsequent investigation often aims at clarifying the general principles or components that mastery of the task involves. Often there is a separation of highly specific task information from more general information. Ernst and Newell's (1969) discussion of the General Problem Solver is a classic example. The general procedures of means-ends analysis were implemented as a distinct program that was run in conjunction with representations of several different problem domains. Similar remarks apply to natural language understanding systems, such as Winograd's (1972) and Woods' (1972), that separated a general syntactic parser from a task specific lexicon and grammar, and a semantic representation language from some task-specific information written in it. Hypotheses that identify some of the components of a theory as being relatively more general than others provide a step in the direction of making a theory principled, in the sense that we propose.

In our view, a significant strengthening of computational theories can be achieved by explicating their principles and the *entailments* of the principles. We believe that a substantial advance is achieved if a theory can be developed beyond being a black box with a certain measured adequacy. This involves laying bare the theory's principles and their entailments, showing how each principle, *in the context of its interactions with the others*, increases empirical coverage, or reduces tailorability, or improves the adequacy of the theory in some other way. With such developments, the theorist provides explanations of

why the particular principles were chosen. The support structure for each principle is laid bare.

The internal structure of the theory—the way the principles interact to entail empirical coverage and tailorability—comes out best when the theory is compared with other theories and with alternative versions of itself. That is, the key to supporting theories appears to be *competitive argumentation*. This style of support has succeeded in certain deep, non-computational theories. We suggest that it can be adapted to the increased rigor and detail of computational theories.

In practice, most competitive arguments have a certain "king of the mountain" form. One shows that a principle accounts for certain facts, and that certain variations or alternatives to the principle, while not without empirical merit, are flawed in some way. That is, the argument shows that its principle stands at the top of a mountain of evidence, then proceeds to knock the competitors down. The second section of this chapter illustrates the notion of such an argument with an example.

Competitive arguments hold promise for establishing which principles are crucial for analyzing cognition. To show that some constraint is crucial is to show that it is *necessary* in order for the theory to meet some criteria of adequacy. To show that it is *sufficient* is not enough. Indeed, any successful theory that uses some principle is a sufficiency argument for that principle. But when there are two theories, one claiming that principle X is sufficient and another claiming that a different, incompatible principle Y, is sufficient, sufficiency itself is no longer persuasive. One must somehow show that X is better than Y. Indeed, this sort of competitive argumentation is the only realistic alternative to necessity arguments. *Competitive arguments form a sort of successive approximation to necessity.*

Argumentation in non-computational fields

Well reasoned, competitive argumentation occurs in non-computational fields. Linguistics is a particularly good example. Throughout its history, linguistics has had a strong empirical tradition. Prior to Chomsky, syntactic theories were rather shallow and almost taxonomic in character. The central concern was to tune a grammar to cover all the sentences in a given corpus. Arguments between alternative grammars could be evaluated by determining which sentences in the corpus could be analyzed by each. When Chomsky reshaped syntax by postulating abstract remote structures, namely a base grammar and transformations, argumentation had to become much more subtle. Because transformations interacted with each other and the base grammar in complex ways, it was difficult to evaluate the empirical impact of alternative formulations of rules.

As Moravcsik has pointed out (Moravcsik, 1980), Chomskyan linguistics is virtually alone among the social sciences in employing deep theories. Moravcsik (1980) labels theories "deep" (without implying any depth in the normative sense) if they "refer to many layers of unobservables in their explanations. . . . 'Shallow' theories are those that try to stick as close to the observables as possible, [and] aim mostly at correlations between observables. . . . The history of the natural sciences like physics, chemistry, and biology is a clear record of the success story of 'deep' theories. . . . When we come to the social sciences, we encounter a strange anomaly. For while there is a lot of talk about aiming to be 'scientific,' one finds in the social sciences a widespread and unargued-for predilection for 'shallow' theories of the mind [p. 28]."

Computational theories of cognition are "deep" theories because much of the mechanism and representation that they postulate is quite unobservable. This depth is another reason that argumentation has been rare in computational theories. The principles and components that structure the theory's computation are remote from the data. The derivation of predictions from them is often so lengthy and convoluted that only by executing the computation on the computer can the theorist tell what the current version of the theory predicts. To assign empirical responsibility to a *component* of the remote structure is possible only in rare cases. The depth of computational theories makes establishing the empirical necessity of their principles or architecture extremely difficult, even given that their sufficiency has been demonstrated.

When theories are "shallow," then argumentation is easy. In a sense, the data do the arguing for you. Most experimental psychology is like this. The arguments are so direct that the only place they can be criticized is at the bottom, where the raw data is interpreted as findings. Experimental design and data analysis techniques are therefore of paramount importance. The reasoning from finding to theory is often short and impeccable. On the other hand, when theories are deep in that the derivation of predictions from remote structures is long and complex, argumentation becomes lengthy and intricate. However, the effort spent in forging them is often repaid when the arguments last longer than the theory. Indeed, each argument is almost a micro-theory. An argument's utility may often last far longer than the utility of the theory it supports. This utility may take the form of a crucial fact, as discussed later.

The transition from studying shallow theories to studying deep ones is sometimes accompanied by something like culture shock. The heightened concern with intricate argumentation from apparently casual empirical observations strikes some as totally unwarranted. Yet this concern can be highly significant, since it can show where the theoretical connections are weakest. It is here that theories draw fire from their critics, and rightfully so, for there are many trails between surface findings and remote structures, and the first one traversed is not always the best.

There is another orientation that can block acceptance of argumentation. Cognitive science often equates the merit of a theory with the complexity of the task domain that it addresses. This is entirely appropriate when the theory consists solely of a task analysis. If the task domain is trivial, an analysis of its

information structure is often trivial, or at least less interesting than an analysis of a complex task. As we suggested earlier, task analyses are and will continue to be an important first step in cognitive studies. They have dominated the field since the 1950s and distinguish cognitive psychology from its more task-information-free predecessors. Their dominance has made it almost inevitable that the complexity of the theory's task domain has been strongly associated with the theory's perceived merit, even if that theory goes well beyond an analysis of its domain. Often, the complexity of argumentation forces the research to be conducted in simple task domains. Since the task analysis is only a first step and a relatively easy one compared to eliciting principles and constructing supporting arguments, simple task domains are a good choice for such research. Moreover, the simplicity of the domain may allow discovery and sharpening of research tools, the cognitive equivalent of the stains of neuroanatomy or the restriction enzymes of microbiology. Such tools, developed in a simple domain, can be used to illuminate a whole area. Yet, choosing a simple domain makes it difficult to attract the attention of the cognitive science community, which often equates simple tasks with trivial theories. Yet this attention is sorely needed, not only to encourage the investigators, but because competitive argumentation, even more so than other forms of support, thrives on challenge. The entirely appropriate habits of the early years of cognitive science seem to be dampening the emergence of principled, well-argued theories.

Crucial facts

Ultimately, every theory is abandoned. In the long run, it might seem that the effort spent on careful argumentation is wasted since argumentation is so thoroughly embedded in the theory's framework. We do not believe this is the case. A long term effect of an argument is to raise the prominence of a certain set of observations, making them "crucial facts," in the linguistic jargon. For example, a previously obscure class of sentences (the "promise-persuade" sentences) became important as supporting data for two transformations of early transformational grammars (Raising and Equi-NP Deletion). A typical pair from this class is

> Ronald promised Margaret to pay the bill.
> Ronald persuaded Margaret to pay the bill.

Promise-persuade data have shaped *every* successor of transformational grammars. They have become, like active/passive sentence pairs, crucial facts that any serious grammar must explain.

Another example comes from behavioral theory in the domain of instrumental conditioning. Hull (1943) formulated a hypothesis in which the strength of a tendency to respond depends on a global motivational factor, drive, and a factor

depending on the organism's experience, habit strength. The habit strength of a response was assumed to depend both on the number of times the response had occurred and been followed by reinforcement, and on the amount of reinforcement received. In an ingenious experiment, Tolman and Honzik (1930) placed rats in a maze and permitted them to move about but provided no apparent reinforcement. Then food was placed in a specific location, and the rats were given learning trials. The rats with exploratory experience learned to go to the food more quickly than rats that had not received the experience. This phenomenon of *latent learning* played an important role in the further development of behavior theory.

Commonly, crucial facts play the role of deciding among two important hypotheses. They could almost be called "decisive" facts for this reason. The two examples above were decisive. Promise-persuade sentences demonstrate that there must be two distinct transformations operating in roughly the same syntactic domain. Although the surface syntax is similar, advocating a single-transformation approach, the apparent differences in which noun phrase is the implicit subject of the subordinate clause (i.e., who pays the bill) shows that the more complex hypothesis of two transformations is necessary. Similarly, the fact of latent learning played a decisive role. It showed that the effects of experience may not be reflected directly in performance. The distinction between learning and performance had to be made more complex. In Hull's theory, latent learning was accommodated by changing the assumption that habit strength grows as a function of the amount of reinforcement. In later versions of the theory (e.g., 1952) Hull assumed that habit strength depends only on the *number* of occurrences of the response that have been followed by reinforcement, and not on the *amount* of reinforcement. To account for the affect of amount of reinforcement, Hull assumed it acts as a motivational factor, called incentive, which influences performance rather than learning. Hull was then able to account for latent learning by arguing that some small amount of reinforcement probably was provided in the exploratory trials—for example, the rat likes going home to its cage, so it takes removal from the maze as reinforcement for its last few moves. Since there was reinforcement, there was learning. But the amount of reinforcement was small, so the incentive to exhibit this learning was small. Hence, the rat's learning was not reflected in performance until food was introduced.

Few, if any, crucial facts have emerged as yet in relation to computational theories of cognition. For example, SHRDLU's famous sentence "Pick up the big red block," is not considered a necessary part of a parser's repertoire, nor must a learner learn to recognize an arch made of blocks, despite the centrality of these examples in Winograd's and Winston's work. The reason no old examples seem worth accounting for in new theories is that no theoretical issues hinged on them in the old theories. They were used for illustration, and not to argue principles.

Arguments and crucial facts often survive longer than theories. Theorists repeatedly appeal to crucial facts, we believe, because they already know quite a few ways of accounting for these facts, most of which are somehow flawed. This makes them convenient tests for a new theory. Although the counter-arguments will most often not lift directly over to the new theory, they will at least hint at where to look for trouble.

We suggest that long term progress may be exactly the accretion of crucial facts, arguments, and possibly even techniques or types of arguments. Theories come and go. Wherein lies our accumulated knowledge? Perhaps the knowledge that separates a "mature" field from a young one is the empirical ramifications and other entailments of its ideas—the arguments connecting them to the facts and to each other.

Competence theories and star data

Recently, efforts have been increasingly directed at formulating theories of what people *can* do rather than what they *did* do in the given situation. In part, this follows from the realization that the performance of subjects when confronted with a task is strongly determined by the requirements of the task. To use Simon's metaphor (Simon, 1981), studying the path of an ant across the beach tells one more about the beach than the ant. To learn something about general principles of cognition (the ant), it is necessary to abstract detailed observations (the path). This introduces a level of inference to that usually encountered when performance is used for inferences about underlying cognitive processes and structures. Founding this new level, of course, involves argumentation.

Another reason for emphasizing competence (*can* do) over performance (*does* do) is that it allows side stepping, to a certain degree, the difficult issue of allowing some tuning of performance parameters around individual differences without introducing unlimited degrees of tailorability into the theory.

Assessing the underlying competence is no easy empirical task. The conclusions one draws can be colored by subtle variations in the task. An illustrative case is the controversy over the development of competence in counting. Gelman's classic "magic" experiments showed that the elements of number competence are present much earlier than strict Piagetian theory would predict (Gelman & Gallistel, 1978).

A successful style of argumentation has emerged in service of competence theories. If one defines a performance theory as a theory of what the subject *does* do, and a competence theory as a theory of what the subject *can* do, then clearly evidence about what the subject *can't* do will be very important for narrowing the range of behavior allowed by a competence theory. Such evidence is called *star data*.

A star datum is a simulated behavior that no human's behavior would ever match. It is named after the lingustic convention of starring sentences that are not

in the language. How one ascertains the non-existence of a behavior may vary in different domains. By definition, star data are behaviors that are not found naturally. Thus, star data can not be observed in the same (potentially) objective ways that ordinary data are observed. In particular, in many domains, a subject can easily perform the behavior when asked to (e.g., utter "Furiously sleep ideas green colorless") even though they would never do so naturally. In current linguistic practice, sentences are starred according to the judgment of native speakers. For Repair Theory (the theory that the forthcoming example is drawn from), expert diagnosticians were the judges of whether the model's behavior represented a star datum or plausible human behavior that just hadn't been observed yet.

Star data can be particularly useful in supporting claims about mental representations. One of the techniques that a theorist can use to structure knowledge is to stipulate that it must be represented in a certain formalism, often called the representation language. A narrowly defined language reduces the ways that a piece of knowledge can be decomposed by forcing its decomposition to fit into the forms allowed by the representation language. This technique raises formalisms from the status of mere notations to bearers of important theoretical claims. Star data can be useful in supporting such claims: If it can be shown that the theory would generate certain star data if it were not constrained by the given representation language, then one has a strong argument for the utility, if not the actual psychological reality of that representation language. If the representation language is extremely successful in constraining knowledge structures, one might even be inclined to propose it as a mental representation (Fodor, 1975).

AN EXAMPLE OF AN ARGUMENT

In this section, we give an example of an argument. However, arguing is impossible in vaccuo, so several paragraphs must be spent in describing the theory the argument is taken from and the data that supports it. Like most arguments, this one depends strongly on assumptions of the theory which themselves have supporting arguments. This leads to rather involved, convoluted reasoning—a characteristic of competitive argumentation exacerbated by the complexities of protocol data.

The argument presented here is taken from VanLehn (1983). It concerns Repair Theory (Brown & VanLehn, 1980). Repair Theory examines certain cognitive aspects of procedural skills. The basic idea of Repair Theory is that while following a procedure students will barge through any trouble that they encounter by doing only a small amount of superficial "patching." That is, they make a minimal change to the procedure's execution state in order to circumvent the trouble and get back on the track. Sometimes they perform different repairs for the same trouble, other times they use the same repair for periods of time, and

sometimes they seem to abstract the patch and make it a part of their procedure. Repair Theory is concerned with formalizing this overall impression so that it can be tested. In doing so, it aims to articulate a formal representation for knowledge about procedures—a *mentalese* for procedures (Fodor, 1975)—and an architecture for interpreting that knowledge and for managing trouble during its interpretation.

The procedural skill taken for study is ordinary multi-column subtraction. The subjects are elementary school students who are in the process of learning that procedure. The main advantage of these choices, from a psychological point of view, is that for these subjects, subtraction is a virtually meaningless procedure. Most elementary school students have only a dim conception of the underlying semantics of subtraction, which are rooted in the base-ten representation of numbers. When compared to the procedures they use to operate vending machines or play games, subtraction is as dry, formal, and disconnected from everyday interests as the nonsense syllables used in early psychological investigations were different from real words. This isolation is the bane of teachers but a boon to the cognitive theorist. It allows one to study a skill formally without bringing in a whole world's worth of associations. This isolation provides an elegant opportunity for building a microscope into the mentalese of procedural knowledge and the architecture for interpreting it.

The data supporting the theories comes from the Buggy studies (Brown & Burton, 1978; VanLehn, 1981). From the errors of thousands of students taking ordinary pencil and paper subtraction tests, a hundred primitive *bugs* were inferred. Bugs are a formal device for notating systematic errors in a compact way. They are designed so that any of the observed systematic errors can be expressed by a set of one or more bugs. (Actually, bugs often do not combine linearly. The co-occurrence of two or more bugs is called a "compound" bug.) This notation is precise in that it describes not only what problems are answered incorrectly, but what the contents of those answers were and what steps were followed in producing them. The technicalities of this form of data have been discussed in other papers (Brown & Burton, 1978; Burton, 1981). The behavioral data that support the arguments will be presented here as ideal protocols of the subject's local problem solving. This simplifies the exposition considerably.

A critical distinction in Repair Theory is between regular execution and *local problem solving*. Regular execution depends of course on what the representation of procedures is. If procedures are represented as pushdown automata, for instance, regular execution is just following arcs, pushing and popping the stack, testing arc conditions, executing primitive arc actions, and so forth. If the procedure is somehow flawed, perhaps because the student mislearned it or forgot part of it, then regular execution may get stuck. When a student executing a procedure reaches an impasse, the student is unlikely to just halt, as a pushdown automaton does when it can't execute any arcs leaving its current state. Instead, the student will do a small amount of problem solving, just enough to get unstuck

and resume regular execution. These local problem solving strategies are called *repairs* despite the fact that they rarely succeed in rectifying the broken procedure, although they do succeed in getting past the impasse. Repairs are quite simple tactics, such as skipping an operation that can't be performed or backing up in the procedure in order to take a different path. Repairs do not in general result in a correct solution to the exercise the procedure is being applied to, but instead result in a buggy solution. The theory explains the large variety of observed subtraction bugs as the result of a few flawed underlying subtraction procedures being subjected to local problem solving involving a surprisingly small set of repairs. Local problem solving is twofold: detecting impasses, called *criticism*, and getting around them, called repair. Some bugs result from several instances of impasse/repair during the application of an underlying flawed procedure to a problem.

There are strong constraints on criticism and repair. Both criticism and repair are very simple and local. Two main types of criticism are detecting when an action's precondition is violated (e.g., trying to decrement a zero) and detecting when regular execution halts because none of its methods are applicable to the current situation. Repairs are also simple and local: for example, skipping an operation or backing up in the procedure. A basic principle of Repair Theory is that *any repair can be applied at any impasse*, subject only to the condition that it succeed in getting the procedure past the impasse. The empirical impact of this principle is that the theory predicts that *the set of all possible bugs is exactly the set of all possible repairs applied to all possible impasses*. To summarize: the student's observed behavior is a combination of regular execution and local problem solving, where local problem solving consists of detecting an impasse and applying one of a set of repairs to it.

Assumptions

Although a great deal more can be said about Repair Theory and bugs, it is time to turn to the illustrative argument. It concerns the architecture/representation that the theory uses to model human conceptions of procedures, and in particular, the component of the architecture that is called the short-term or working memory in production systems. This argument is part of a longer argument that the representation should be an applicative language. The principle at issue here is how the model should represent focus of attention. As students solve a subtraction problem, they focus their attention on various digits or columns of digits at various times. This can be inferred from the information that students read from the paper, or perhaps from eye-tracking studies. For this chapter, it will be assumed that focus and focus shifting exist. The issue to discuss is how to represent them. A leading contender will be a "you are here" register that stores the focus, that is, where on the paper the attention is being focused. This register, in tandem with a "currently active goal" register, puts a clean division between

focus of attention and control state. However, register-based architectures will be shown to make focus *too* independent of control state. A different architecture, one that unites focus and control, will be shown to fit the facts more closely. It's named schema/instance for reasons that will become clear in a moment.

We would like to speak in an informal way of goals and subgoals, intending that these be taken as referring to the procedural knowledge of subtraction itself, rather than expressions in some particular representation (e.g., production systems, and-or graphs, etc.). In particular, we'll assume that borrowing is a subgoal of the goal of processing a column, and that borrowing has two subgoals, namely borrowing-from and borrowing-into. Borrowing-into is performed by simply adding ten to a certain digit in the top row. The borrowing-from subgoal is realized either by decrementing a certain digit, or by invoking yet another subgoal, borrowing-across-zero. (This way of borrowing is not the only one, but it was the one taught to all our subjects.) These assumptions, or at least some assumptions, are necessary to begin the discussions. They are, we believe, some of the mildest assumptions one can make and still have some ground to launch from. We've assumed that focus exists and that a goal-structured control regime exists. One other assumption is needed before the main argument can be presented. We'll assume that a repair called *Backup* exists. This repair is most easily described with an example of its operation.

The Backup repair in action

Figure 9.1 gives an idealized protocol. It illustrates a moderately common bug (Smaller From Larger Instead of Borrow From Zero). In a sample of 417 students with bugs, five students had this bug (VanLehn, 1981). The (idealized) subject of Fig. 9.1 does not know all of the subtraction procedure. In particular, he does not know about borrowing from zero. When he tackles the problem 305–167, he begins by invoking a process-column goal. Since 5 is less than 7, he invokes a borrow subgoal (episode *a*, see Fig. 9.1), and immediately the first of borrowing's two subgoals, namely borrowing-from (episode *b*). At this point, he gets stuck because the digit to be borrowed from is a zero, which can not be decremented in the natural number system. In Repair Theoretic terms, he has reached an *impasse*.

The theory describes several *repair strategies* that can be used at impasses to get unstuck. The one that interests us here is the Backup repair. It gets past the decrement-zero impasse by "backing up," in the problem solving sense, to the last goal that has some open alternatives. In this case, there are four goals active:

borrowing-from
borrowing
processing a column
solving the subtraction problem

a. 305 − 167 — In the units column, I can't take 7 from 5, so I'll have to borrow.

b. 305 − 167 — To borrow, I first have to decrement the next column's top digit. But I can't take 1 from 0!

c. 305 − 167 = 2 — So I'll go back to doing the units column. I still can't take 7 from 5, so I'll take 5 from 7 instead.

d. 2 1 / 3 0 5 − 167 = 2 — In the tens column, I can't take 6 from 0, so I'll have to borrow. I decrement 3 to 2 and add 10 to 0. That's no problem.

e. 2 1 / 3 0 5 − 167 = 142 — Six from 10 is 4. That finishes the tens. The hundreds is easy, there's no need to borrow, and 1 from 2 is 1.

FIG. 9.1 An idealized protocol of a student with the bug Smaller From Larger Instead of Borrow From Zero.

The borrowing-from goal has failed. The borrow goal has no alternatives: one always borrows-from then borrows-into. The next most distant goal, namely column-processing, has alternatives: one for columns that need a borrow, one for columns that do not need a borrow. So Backup returns control to the column-processing goal. Evidence for backing up occurs in episode *c*, where the subject says "So I'll go back to doing the units column." In the units column he hits a second impasse, saying "I still can't take 7 from 5," which he repairs ("so I'll take 5 from 7 instead"). He finishes up the rest of the problem without difficulty.

The crucial feature of the analysis above, for this argument, is that Backup caused a transition from a goal (borrowing-from) located at the top digit in the *tens* column to a prior goal (processing a column) located at the *units* column. Backup caused a shift in the focus of attention from one location to another as well as from one goal to another. Moreover, it happens that the location it shifted back to was the one that the process-column goal was originally invoked on, even though that column turned out to cause problems in that further processing of it led to a second impasse. So, it seems no accident that Backup shifted the location back to the goal's original site of invocation. It is because Backup shifts

both *focus* and *control* that it is the preeminent tool to be used in the argument that follows.

Overview

The issue is how to represent focus. To keep the argument short, just two alternative architectures are discussed: register-based and schema/instance. As it turns out, just two facts are needed. One is the bug just described. The other will be introduced in a moment. The argument is organized around these two facts. Figure 9.2 is an outline of the argument.

It will be shown that the schema/instance architecture generates the first bug (I.A in the outline), but the simplest version of the register-based architecture, a single focus register, generates a star bug instead (I.B.1). Patching its difficulties by making the Backup repair more complex leads to problems with retaining the falsifiability of the theory (I.B.2). However, using several registers instead of just one register allows the bug to be generated simply (I.B.3). So the conclusion to be drawn from the first fact will be that the register approach will be adequate only if there is more than one focus register.

The second part of the argument introduces a new bug involving the Backup repair. Once again, the schema/instance architecture predicts the fact correctly (II.A). Two different implementations of multiple registers fail (II.B.1 and II.B.2) by generating star bugs. Smart Backup would fix the problem but remains methodologically undesirable (II.B.3). Postulating various complications to the goal structure of the procedure (II.B.4 and II.B.5) allow the correct

I. Fact 1: Smaller From Larger Instead of Borrow From Zero
   A. The schema/instance architecture generates it
   B. Registers
      1. A single register architecture generates a star bug
      2. "Smart" Backup is irrefutable, hence rejected on methodological grounds
      3. Multiple register architecture allows generation of SFLIBFZ

II. Fact 2: Borrow Across Zero
   A. The schema/instance architecture generates it
   B. Registers
      1. One register per goal: can't generate the bug
      2. One register per object: can't generate the bug
      3. "Smart" Backup is irrefutable, hence rejected on methodological grounds
      4. Duplicate borrow-from goals: entails infinite procedure
      5. Duplicate borrow goals: equivalent to schema/instance

FIG. 9.2 Outline of the argument.

predictions to be generated, but they have problems of their own. So the second part concludes that the register-based alternative is inadequate even when various complex versions of it are introduced.

In overview, the argument is a nested argument-by-cases where all the cases except one are eliminated. To aid in following it, the cases are labelled as they are in the outline of Fig. 9.2.

I.A Schema/instance generates the bug

The basic idea of a schema/instance architecture is that the location of a goal is very strongly associated with the goal at the time it is first set. That is, when a goal like borrowing is invoked, it is invoked at a certain column, or more generally at a certain physical location in the visual display of the subtraction problem. In a schema/instance architecture, this association between goal and location, which is formed at invocation time, persists as long as the goal remains relevant. That is, the goal is a *schema*, which is *instantiated* by substituting specific locations, numbers, or other data into it. Most modern computer languages, such as LISP, have a schema/instance architecture: a function is instantiated by binding its arguments when it is called, and its arguments retain their bindings as long as the function is on the stack.

The schema/instance architecture allows the bug of Fig. 9.1 to be generated quite naturally. Suppose the process-column goal were strongly associated with its location, namely the units column, in some short-term memory. Then Backup causes the resumption of the goal at the stored location. Another way to think of this is that the interpreter is maintaining a short-term "history list" that temporarily stores the various invocations of goals *with their locations*. In regular execution, when the borrowing goal finishes, the process-column goal is resumed *at the same place as it started*. That is, in the long-term representation of the procedure, the process-column goal is a schema with its location abstracted out. It is bound to a location (instantiated) when it is invoked. It is the instantiated goal that Backup returns to, not the schematic one.

This schema/instance distinction, which is at the heart of almost all modern programming languages, entails the existence of some kind of short-term memory to store the instantiations of goals, and thus motivates this way of implementing the Backup repair. But there are of course other ways to account for focus shifting during Backup. Several will be examined and shown to have fewer advantages than the schema/instance one.

I.B.1 A single register architecture generates a star bug

Suppose that instead of using the schema instances to implement focus storage, the architecture used a single register, a "you are here" pointer to some place in the problem array. There would be no problem representing the subtrac-

tion procedure in such an architecture. In order to shift focus left, for example, an explicit action in the procedure's representation would change the contents of this register as the various goals were invoked.

Suppose first off that Backup is kept as simple as possible, and in particular, that it doesn't go rooting around in the procedure in order to find out how to reset the register. Under this parsimonious account of Backup, the single register architecture causes trouble. Because the "you are here" register is simply left alone during backup, a star bug is generated. It is illustrated in Fig. 9.3. At episode *b*, Backup resumes the process-column goal, but the "you are here" register is not restored to the units column. Instead, the tens column is processed. The units column is left with no answer despite the fact that its top digit has been incremented. In the judgment of several expert diagnosticians, this behavior would never be observed among subtraction students. It is a *star bug*. The theory should not predict its occurrence.

I.B.2 Smart Backup repair makes the theory too tailorable

To avoid the star bug, the Backup repair would have to employ an explicit action to restore the register to the units column in episode *c* of the protocol of Fig. 9.1. But how would it know to do this? Backup would have to determine that the focus of attention should be shifted rightwards by doing an analysis of the goal structure contained in the stored knowledge about the procedure. It would see that in normal execution, a locative focus shifting function was ex-

a. $\begin{array}{r} {\scriptstyle 1}\\ 402\\ -106\\ \hline \end{array}$ In the units column, I can't take 6 from 2, so I'll have to borrow. First I'll add ten to the 2.

b. $\begin{array}{r} {\scriptstyle 1}\\ 402\\ -106\\ \hline \end{array}$ I'm supposed to decrement the top zero, but I can't! So I guess I'll back up to processing the column.

c. $\begin{array}{r} {\scriptstyle 1}\\ 402\\ -106\\ \hline 0 \end{array}$ Processing it is easy: 0−0 is 0.

d. $\begin{array}{r} {\scriptstyle 1}\\ 402\\ -106\\ \hline 30 \end{array}$ The hundreds is also easy. I'm done!

FIG. 9.3 An idealized "protocol" for a star bug.

ecuted between the borrow-from goal and the process-column goal. For some reason, it decides to execute the *inverse* of this shift as it transfers control between the two goals.

Not only does this implementation make unmotivated assumptions, but it grants Backup *the power to do static analyses of control structure*. This would give it significantly more power than the other repair heuristics, which do simple, local things like skipping an operation, or executing it on slightly different locations. It gives the local problem solver so much power that one could "explain" virtually any behavior by cramming the explanation into the black boxes that are repair heuristics. That is, allowing repairs to do static analyses gives the theory too much tailorability. It is much better to make the heuristics as simple as possible by embedding them in just the right architecture.

I.B.3 Multiple register allow generation of the bug

Another way to implement Backup involves using a *set* of registers. The registers have some designated semantics, such as "most recently referenced column" or "most recently referenced digit." That is, the registers could be associated with the type or visual shape of the locations referenced (e.g., as Smalltalk's class variables are). Alternatively, they could be associated with the schematic goals. (Some Fortran compilers implement a subroutine's local variables this way by allocating their storage in the compiled code, generally right before the subroutine's entry point.) Process-column would have a register, borrow would have a different register, and so on.

Given this architecture, Backup is quite simple. Returning to process-column requires no locative focus shifting on its part. Since the process-column register (or the column register, if that's the semantics) was not changed by the call to borrow, it is still pointing at the units column when Backup causes control to return to process-column. This multi-register implementation is competitive with the schema/instantiation one as far as its explanatory power (i.e., Backup is simple and local, and the architecture has motivation independent of the Backup repair in that is used during normal interpretation). However, it fails to account for certain empirical facts that will now be exposed.

II.A Another bug, and schema/instance can generate it

The argument in this section is similar to the one in the previous section. It takes advantage of subtraction's recursive borrowing to exhibit Backup occurring in a context where there are two invocations of the borrow goal active at the same time. This means there are two potential destinations for Backup. It will be shown that the schema/instance mechanism is necessary to make the empirically correct prediction.

A common bug is one that forgets to change the zero when borrowing across zero. This leads to answers like:

```
  2
  3
  4'0'2
-1 3 9
-------
  1 7 3
```

(The small numbers stand for student's scratch marks.) The 4 was decremented once due to the borrow originating in the units column, and then again due to a borrow originating from the tens column because the tens column was not changed during the first borrow as it should have been. This bug is called Borrow Across Zero. It is a common bug. Of 417 students with bugs, 51 had this bug (VanLehn, 1981).

An important fact is seen in Fig. 9.4. The bug decrements the 1 to zero during the first borrow. Thus, when it comes to borrow a second time, it finds a zero where the one was, and performs a recursive invocation of the borrow goal. This causes an attempt to decrement in the thousands columns, which is blank. An impasse occurs. The answer shown in the figure is generated by assuming the impasse is repaired with Backup. This sends control back to the most recently invoked goal that has alternatives. At this point the active goals are:

borrow-from	(the recursive invocation located at the thousands column)
borrow-from-zero	(at the hundreds column)
borrow-from	(at the hundreds column)
borrow	(at the tens column)
process-column	(at the tens column)

In this procedure, the borrow-from-zero goal has no alternatives (it should always both write a nine over the zero and borrow-from the next column, although here the write-nine step has been forgotten). The borrow-from goal has alternatives because it has to choose between ordinary, non-zero borrowing and borrowing from zeros. Since borrow-from was the most recently invoked goal that has alternatives left, Backup returns to it. Execution resumes by taking its other alternative, the one that was not taken the first time. Hence, an attempt is made to do an ordinary borrow-from, namely a decrement. Crucially, this happens in the hundreds column, which has a zero in the top, which causes a new impasse. We see that it is the hundreds column that was returned to because the impasse was repaired by substituting an increment for the blocked decrement, causing the zero in the hundreds column to be changed to a one.

The crucial fact is that the Backup repair shifted the focus from the thousands column to the hundreds column, even though both the source and the destination of the backing up were borrow-from goals. This shift is predicted by the schema/instance architecture. However, the empirical adequacy of the register architecture is not as high.

II.B.1 One register per goal can't generate the bug

Suppose each schematic goal has its own register. Borrow-from would have a register, and it would be set to the top digit of the thousands column at the first impasse (episode *b* in Fig. 9.4). Hence, if Backup returns to the first invocation of borrow-from, the register will remain set at the thousands column. Hence, Backup doesn't generate the observed bug of Fig. 9.4. In fact, it can't generate it at all: the only register focused on the hundreds column is the one belonging to borrow-from-zero. That goal has no open alternatives, so Backup can't return to it. Even if it did, it wouldn't generate the bug of Fig. 9.4. So one register per goal is an architecture that is not observationally adequate.

II.B.2 One register per object doesn't generate the bug

Assuming the registers are associated with object types fails for similar reasons. Both the impasses (episodes *b* and *d*) involve the same type of visual

a.
```
  0 1
  1'02
-  39
-----
   3
```
Since I can't take 9 from 12, I'll borrow. The next column is 0, so I'll decrement the 1, then add 10 to the 2. Now I've got 12 take away 9, which is 3.

b.
```
  0 1
  1'02
-  39
-----
   3
```
Since I can't take 3 from 0, I'll borrow. The next digit is 0, but there isn't a digit after that!

c.
```
  0 1
  1'02
-  39
-----
   3
```
I guess I could quit, but I'll go back to see if I can fix things up. Maybe I made a mistake in skipping over that 0, so I'll go back there.

d.
```
  1
  0 1
  1'02
-  39
-----
   3
```
When I go back there, I'm still stuck because I can't take 1 from 0. I'll just add instead.

e.
```
  1
  0 1
  1'02
-  39
-----
  173
```
Now I'm okay. I'll finish the borrow by adding 10 to the ten's column, and 3 from 10 is 7. The hundreds is easy, I just bring down the 1. Done!

FIG. 9.4 An idealized protocol of a student with aversion of the bug Borrow Across Zero.

object, a digit, and hence the corresponding register would have to be reset explicitly by Backup in order to cause the observed focus shift.

II.B.3 Smart Backup makes the theory too tailorable

But providing Backup with an ability to explicitly reset registers would once again require it to do static analysis of control structure—an increase in power that should not be granted to repairs.

II.B.4 Duplicate borrow-from goals

One could object that we have made a tacit assumption that it is the same (schematic) borrow-from that is called both times. If there were two schematic borrow-froms, one for an adjacent borrow, and one for a borrow two columns away from the column originating the borrow, then they could have separate registers. This would allow Backup to be trivial once more. However, this argument entails either that one have a subtraction procedure of infinite size, or that there be some limit on the number of columns away from the originating column that the procedure can handle during borrowing. Both conclusions are implausible.

II.B.5 Duplicate borrow goals

But one could object that there is another way to salvage the multiple register architecture. Suppose that the schematic procedure is extended by duplicating borrow goals (plus registers) as needed. The bug could be generated, but this amounts either to a disguised version of schemata and instantiations, or an appeal to some powerful problem solver (which then has to be explained lest the theory lapse into infinite tailorability). So, this alternative is not really tenable either.

This rather lengthy argument concludes with the schema/instance architecture the only one left standing. What this means is that representations that do not employ the schemata and instances, such as finite state machines with registers or flow charts, can be dropped from consideration. This puts us, roughly speaking, on the familiar ground of "modern" representation languages for procedures, such as stack-based languages, certain varieties of production systems, certain message passing languages, and so on.

The example illustrates the main points

The preceding argument illustrates several of the main points of this chapter. The structure of the argument was to first establish a need, in this case for a data flow scheme, then to examine several alternative architectures that meet the need. This pattern of establishing a need and examining alternatives is characteristic of competitive argumentation.

The argument introduced a crucial fact: Whenever problem solving backs up

to a previously invoked goal, the goal is resumed with the same instantiating information that it had during its original invocation. This was illustrated with two bugs (Fig. 9.1 and Fig. 9.4). We can vouch for the truth of this observation in the local problem solving that accompanies subtraction performances, and we expect it to remain uncontradicted by evidence from other domains. In Newell and Simon's classic study of eye movements during the solution of cryptarithmetic puzzles, for example, there is ample evidence that backing up restores not only the goal, but the focus of visual attention that was current when the goal was last active (Newell & Simon, 1972, pp. 323–325).

The arguments turned mostly on limiting the tailorability of the theory and on avoiding star bugs. Since all the competing explanations for the crucial fact were able to account for it one way or another, it was their entailments that decided their relative merits. Sometimes the machinations necessary to account for the facts would introduce so much power into the theory that it could trivially account for any data; in these cases, the hypothesis was rejected as introducing so much tailorability as to make the theory irrefutable. In other cases, the hypothesis entailed the generation of certain absurd behaviors: star bugs. Such generations were treated like the generation of false predictions despite the fact that empirical claims about existence can never be proven false.

The emphasis on what people *can* and *can't* do as opposed to what they *do* do is apparent in the use of bugs rather than raw protocols as the data supporting the argument earlier. Bugs are an idealization of human behavior. They describe systematic errors and leave aside unsystematic errors (i.e., *slips* in the sense of Norman, 1981) such as $7 - 5 = 3$. Also, bugs isolate distinct behaviors: The bug Smaller From Larger Instead of Borrow From Zero occurred five times in our sample but always in combination with various other bugs (VanLehn, 1981). It never occurred alone. There is a layer of inference involved in this idealization of the raw data that has not been presented here (Burton, 1981; VanLehn, 1981). In addition to these difficult sorts of idealization, there are simple ones such as choosing not to collect timing data or self-report data from subjects as they solved problems. Although the objective of Repair Theory is in part to understand the processes involved in applying procedures to problems, it can be successfully approached, we believe, within a competence theory framework.

Lastly, the argument illustrates what we mean for a theory to have "depth." This attribute correlates with the length and complexity of the theory's arguments. In Repair Theory there are multiple layers—protocols, bugs, interpreter state, and knowledge structures. There are precise relationships between these layers. The format of the structures at each layer as well as the nature of the relationships between them require supporting argumentation to show not only that the proposed architectures are sufficient to account for certain crucial facts, but also that they are the leading edge of a convergence upon empirical necessity in that a careful drawing out of the entailments of competing proposals reveals that each of the competitors is flawed.

AN EXAMPLE OF THE PROGRESSION TOWARD COMPETITIVE ARGUMENTATION

It was suggested earlier that cognitive science research is following a natural progression that is entailed by its emphasis on precise and detailed use of task-specific information. That emphasis necessitates an early phase where information latent in a new task domain is uncovered, often by creating a rough computer simulation of the task behavior. These early formulations become refined as important principles and components are separated from the more task-idiosyncratic information. These are put forward as a sufficient formulation of domain knowledge and skill. From these first articulations of principles, attention naturally turns to supporting the principles and/or revising them. From competition among various analyses of the task domain, a successive approximation of what principles and components are necessary emerges. It was toward this later stage that the bulk of this chapter was addressed. Yet, it seems appropriate to end by showing, again with an example, how this phase of competitive argumentation arises naturally from those that must precede it.

A domain that illustrates this natural progression is young children's understanding of principles of number and quantity. Until recently, based largely on Piaget's seminal investigations, most developmental psychologists accepted the conclusion that significant conceptual competence (in the sense of general ability) regarding number is not achieved until children are about seven or eight years old. Yet, children are able to count sets of objects well before they enter school at four or five years old. On the standard view, children's ability to count objects reflects a procedural knowledge of a rote, mechanical nature, rather than understanding. Changing the definition of "number competence" to exclude counting, which develops too early and hence offers a counterexample to classical stage theory, is exactly what Lakatos (1976) would call "monster barring." Just as Lakatos would predict, this caused considerable attention to be focused on counting.

Observations by Gelman and Gallistell (1978), and others, provided evidence that significant conceptual competence underlies preschool children's performance in counting tasks. One example of such evidence is the observation that although many preschoolers use idiosyncratic lists when they count, these lists are used consistently. For example, a child might count with the list, "one, two, three, six, ten." The consistency of use of such lists is taken as evidence that children appreciate the need for a set of symbols with stable order, and that while they can sometimes acquire the wrong set, the principle of stable order is part of their conceptual competence.

Another example involves performance in tasks where children count objects with an additional constraint superimposed on the counting task. In a typical experiment, the child is asked to count five objects arranged in a straight line, then the experimenter adds a constraint by saying, "Now count them again, but

make this the *one*," pointing to the second object in the line. Most five-year-old children make completely appropriate adjustments of their counting procedures in order to accommodate these constraints. Even children whose modified counting is not completely appropriate still perform in ways that preserve some of the constraints of counting. Their performance provides quite strong evidence that counting reflects a good understanding of number, and is not a rote, mechanical procedure, because they generate novel procedures that conform to some, but not all, of the principles of counting.

Given these results, it became untenable to bar counting from theories of the development of cognitive competencies. One response has been to construct a computational model for the development of number competency. This is where our example of the natural progression of computational analyses of cognition begins.

An analysis of children's counting was conducted by Greeno, Mary Riley, and Rochel Gelman (forthcoming). First, a process model was formulated that simulates salient aspects of children's performance on a variety of counting tasks. This was necessary just to come to grips with the latent information that the tasks required. Next, an analysis was developed in which the procedures in the process model were derived from premises that correspond to certain principles of counting. These premises formalized and extended a particular decomposition of counting competence proposed by Gelman and Gallistel (1978). The steps of the derivation involved use of planning rules for including procedural components in the counting procedure. The outcome of the analysis was a planning net (VanLehn & Brown, 1980) that showed how the various components of the counting procedure are formally related to the counting principles. The result of this phase of the research was to articulate clearly the counting principles, indeed to formalize them as operable rules, and to show that they were sufficient to generate the kinds of counting performances observed by Gelman and others.

From a clear articulation and a first demonstration of sufficiency, the research has begun to focus on supporting these particular principles in various ways. To argue that the particular decomposition of competence chosen is correct, we have been investigating how removing certain principles from the set while adding constraints corresponding to the experimenter's request to "Make that one be *one*," results in the derivation of procedures corresponding to the various counting performances observed under the stressed conditions. If successful, this demonstration could support the particular set of counting principles, and rule out others that are (hypothetically) equally successful at generating counting procedure in unconstrained situations. Thus begins a phase of competitive argumentation, following on naturally after a first formulation of a principled, detailed and precise analysis of counting competence.

This example of natural progression also illustrates how experimental facts can become crucial. The fact that performances of children's counting degrade

along the lines predicted by a certain set of principles is used to decide between that set of principles rather than some others. It also appears to preclude an explanation of counting as some indecomposable, "rote" procedure. Because these experiments have been used to decide important theoretical issues, we expect them to remain crucial facts.

CONCLUSIONS

The development of models that simulate the processing of specific information in detail has required large investments of time in developing tools for model building as well as obtaining a working understanding of the power and limitations of computational concepts and theoretical methodology. Hence, many computational theories have lacked explicit principles, many have not used data, and virtually none have the argumentative support that we have discussed in this chapter. During the period of this early development, it has been inevitable and perhaps even desirable that computational theories should have been relatively unprincipled and unsupported. However, the rapid initial growth in computational experience, understanding and tool building seems to be leveling out now. We suggest that it is time to use that investment to initiate a parallel growth in our understanding of what constitutes principled computational theories of mind and what tools would facilitate their construction and especially their support.

Some kind of defense of individual theoretical principles, whether competitive or not, seems necessary to expedite scientific progress. If the support for a theory is not analyzed so that one can see how the evidence bears on each part, then the theory must be accepted or rejected as a whole. In contrast, argumentation allows the theory to be revised incrementally. Indeed, perhaps it is because of the infrequent use of argumentation in cognitive science that its theories "have stood on the toes of their predecessors, rather than their shoulders" (Bobrow, 1973).

Competitive argumentation can have several advantages. In addition to its function of showing the lack of support for some theoretical principles while favoring other principles, it adds information at a more general level, enriching the understanding of the connection between facts and abstractions. A second potential advantage is that by making explicit the reasons for rejecting a principle, when future development of the theory brings the chosen principle into conflict with others or into conflict with new facts, one can sometimes dust off one of the fallen competitors and patch its flaws, rather than searching for a replacement from scratch. In this respect, an argument functions like the "support" assertions that must be saved in order to do dependency-directed backtracking (c.f., de Kleer & Doyle, 1981; Stallman & Sussman, 1977). In short, the criticism of alternative explanations that a typical argument provides is value added to the demonstration of empirical support for the chosen principle. Third,

contrasting two alternative explanations sharpens both, making it easier to understand the positions involved by explicating the considerations that mutually support them as well as those that distinguish them. Fourth, argumentation guards against reinvention of the wheel, for if no argument can be found to split two proposals, one begins to suspect that they are equivalent in all but name. Finally, there is the possibility that arguments will outlive the theory they were crafted to support. They might survive not only as crucial facts, but also perhaps as argumentative techniques. Some of the most significant contributions to mathematics have been innovative proof techniques, techniques that have far outshown the theorems they supported. Perhaps cognitive science will also evolve a repertoire of argumentative techniques.

ACKNOWLEDGMENTS

We would like to thank those whose have helped us with their criticisms of early drafts: Danny Bobrow, Johan de Kleer, Bob Lindsay, Mark Stefik, Bonny Webber, and the editors. This research was supported in part by grants N00014-82-C-0067 and N00014-79-C-0215 under contract authority identification numbers NR667-477 and NR667-430 from the Office of Naval Research, Personnel and Training Research program.

REFERENCES

Bobrow, D. G. Natural language input for a computer problem-solving system. In M. L. Minsky (Ed.), *Semantic information processing*. Cambridge, Mass.: MIT Press, 1968.

Bobrow, D. G. Address given at Third Annual International Joint Conference in Artificial Intelligence, Stanford University, Stanford, Calif., 1973.

Brown, J. S., & Burton, R. B. Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 1978, *2*, 155-192.

Brown, J. S., & VanLehn, K. Repair Theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 1980, *4*, 379-426.

Burton, R. B. DEBUGGY: Diagnosis of errors in basic mathematical skills. In D. H. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems*. London: Academic Press, 1981.

de Kleer, J., & Doyle, J. Dependencies and assumptions. In A. Barr & E. Feigenbaum (Eds.), *The AI handbook*, Los Altos, Calif., Kaufmann, 1981.

Dresher, B. E., & Hornstein, N. On some supposed contributions of artificial intelligence to the scientific study of language. *Cognition*, 1976, *4*, 321-398.

Dreyfus, H. L. *What computers can't do*. New York: Harper & Row, 1972.

Ernst, G. W., & Newell, A. *GPS: A case study in generality and problem solving*. New York: Academic Press, 1969.

Evans, T. G. A heuristic program to solve geometric analogy problems. In M. Minsky (Ed.), *Semantic information processing*. Cambridge, Mass.: MIT Press, 1968.

Fodor, J. A. *The language of thought*. New York: Crowell, 1975.

Fodor, J. A. *Representations: Philosophical essays on the foundations of cognitive science*. Cambridge, Mass.: MIT Press, 1981.

Gelman, R., & Gallistel, C. R. *The child's understanding of number*. Cambridge, Mass.: Harvard University Press, 1978.

Greeno, J. G., Riley, M. S., & Gelman, R. *Conceptual competence and young children's counting*. Unpublished manuscript, forthcoming.

Hull, C. H. *Principles of behavior*. New York: Appleton-Century, 1943.

Hull, C. H. *A behavior system*. New Haven: Yale University Press, 1952.

Kaplan, R. M. *A competence based theory of psycholinguistic performance*. Psychology colloquium at Stanford University, May 1981.

Lakatos, I. *Proofs and refutations: The logic of mathematical discovery*. Cambridge, England: Cambridge University Press, 1976.

Marr, D. Early processing of visual information. *Philosophical Transactions of the Royal Society, B*, 1976, *275*, 483–524.

Marr, D., & Nishihara, H. K. Visual information processing: Artificial intelligence and the sensorium of sight. *Technology Review*, 1978, *8*, 2–23.

Moravcsik, J. M. Chomsky's radical break with modern tradition (A commentary on Chomsky's Rules and Representations). In N. Chomsky, Rules and representations. *The Behavioral and Brain Sciences*, 1980, *3*, 1–63.

Newell, A., & Simon, H. A. *Human problem solving*. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Newell, A., Shaw, J. C., & Simon, H. A. Empirical explorations with the logic theory machine: A case study in heuristics. In E. A. Feigenbaum & J. A. Feldman (Eds.), *Computers and thought*. New York: McGraw-Hill, 1963.

Norman, D. A. Categorization of action slips. *Psychological Review*, 1981, *88*, 1–15.

Pylyshyn, Z. W. Computation and cognition: Issues in the foundations of cognitive science. *The Behavioral and Brain Sciences*, 1980, *3*, 111–168.

Simon, H. A. *Sciences of the artificial*. Second edition. Cambridge, Mass.: MIT Press, 1981.

Simon, H. A., & Kotovsky, K. Human acquisition of concepts for sequential patterns. *Psychological Review*, 1963, *70*, 534–546.

Stallman, R. M., & Sussman, G. J. Forward reasoning and dependency-direct backtracking in a system for computer aided circuit analysis. *Artificial Intelligence*, 1977, *9*, 135–196.

Tolman, E. C., & Honzik, C. H. Introduction and removal of reward and maze performance in rats. *University of California Publications in Psychology*, 1930, *4*, 257–275.

VanLehn, K. *Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills* (Tech. Rep. CIS-11). Palo Alto, Calif.: Xerox Palo Alto Research Centers, 1981.

VanLehn, K. On the representation of procedures in Repair Theory. In H. Ginsberg (Ed.), *The development of mathematical thinking*. New York: Academic Press, 1983.

VanLehn, K., & Brown, J. S. Planning Nets: A representation for formalizing analogies and semantic models of procedural skills. In R. E. Snow, P. A. Federico, & W. E. Montague (Eds.), *Aptitude, learning and instruction: Cognitive process analyses of learning and problem solving*. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1980.

Vere, S. A. *Induction of concepts in the predicate calculus*. Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR, 1975, 281–287.

Vere, S. A. Inductive learning of relational productions. In D. A. Waterman & F. Hayes-Roth (Eds.), *Pattern directed inference systems*. New York: Academic Press, 1978.

Winograd, T. *Understanding natural language*. New York: Academic Press, 1972.

Winograd, T. On some contested suppositions of generative linguistics about the scientific study of language. *Cognition*, 1977, *5*, 151–179.

Winston, P. H. Learning structural descriptions for examples. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill, 1975.

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