CHAPTER 7

Transfer of Cognitive Skills

WAYNE D. GRAY
JUDITH M. ORASANU

I. INTRODUCTION

The cognitive science revolution (Baars, 1986; Gardner, 1985) has made impressive advances in our understanding of cognitive performance and the acquisition of cognitive skills. Based upon these successes powerful theories are emerging that can predict when and how much transfer will occur. While these theories are diverse, their common denominator is the problem space hypothesis (Newell & Simon, 1972), with many of these theories using production systems (Brownston, Farrell, Kant, & Martin, 1985) as a tool for theory development and representation.

In this chapter we discuss the transfer of complex cognitive skills, such as text editing, with an emphasis on the relationship between transfer and learning. In this section we introduce our topic by comparing the cognitive science approach to transfer with the older verbal learning tradition. We then provide a short discussion of the central facts of transfer and finish the section by considering a number of cognitive science concepts and methods that are importannt to the new study of transfer.

A. A BRIEF COMPARISON OF THE VERBAL LEARNING AND THE COGNITIVE SCIENCE APPROACHES TO TRANSFER

To compare the verbal learning and cognitive science approaches to transfer, we must introduce two key concepts: the distinction between general (or nonspecific) and specific transfer (e.g., McGeoch, 1942; Postman, 1971) and the cognitive science distinction between declarative and procedural knowledge (Winograd, 1975).

1This chapter represents the views of the authors and does not reflect Army policy.

Copyright © 1987 by Academic Press, Inc. All rights of reproduction in any form reserved.
Specific transfer refers to the identity or similarity of elements in the transfer task to elements in the prior task(s). According to Thorndike (1914; Thorndike & Woodworth, 1901), positive transfer is a function of the number of elements in common between two tasks. General or nonspecific transfer refers to the carry-over of attitudes, methods, or techniques of attack (Garrett, 1941), or what McGeoch (1942) referred to as general principles, modes of attack, or set. Such theories of transfer by generalization (Allport, 1937; Judd, 1908) were proposed as alternatives to Thorndike's identical elements position. However, by the early 1940s, these conflicting theories were viewed as different aspects of the same underlying phenomena, the main difference being one of emphasis (Garrett, 1941; McGeoch, 1942).

It is clear that by the 1950s and 1960s the mainstream of transfer research, the verbal learning tradition, no longer viewed general and specific as competing theories, but rather as separate components. Most theory development focused on specific transfer. Here the common elements approach carried the day. The theoretical controversies concerned the mechanisms by which various common elements facilitated or interfered with the acquisition or retention (Postman & Underwood, 1973) of other common elements.

In contrast, general transfer was most typically viewed as something to control for in experiments on specific transfer (as witnessed by the popularity of the C-D control group in studies of paired-associate transfer or retention, see Postman, 1971). When general transfer was studied directly, it was not to resolve a theoretical dispute, but more commonly to describe basic empirical relationships such as whether experience with the method of practice or class of material yielded greater positive transfer (Postman & Schwartz, 1964).

With the emphasis on empirical description, theoretical discussions of the basis of general transfer remained at the common something level. For example, in discussing the general transfer phenomena of learning-to-learn, Postman, Keppel, and Zacks (1968) concluded that it "reflects circumscribed habits and skills determined by the particular conditions of training and capable of varying degrees of generalization to new learning situations." It is in defining this common something that the cognitive science distinction between declarative and procedural knowledge (Winograd, 1975) becomes important.

Declarative knowledge is knowledge of facts or static knowledge (e.g., in what year did Neil Armstrong walk on the moon?), whereas procedural knowledge is knowledge of how to process or manipulate information to accomplish a given task (such as required to do multicolumn multiplication). Models of cognitive skill performance (Newell & Simon, 1972) use this procedural/declarative distinction together with the problem space hypothesis (see section I.O) to represent the knowledge involved in tasks such as solving cryptarithmetic puzzles (Newell & Simon, 1972), physics problems (Larkin, McDermott, Simon, & Simon, 1980), and text editing (Card, Moran, &
Newell, 1980). These tasks have been analyzed into elements of procedural knowledge such as goals and subgoals, methods, selection rules, and operators which either incorporate or act upon declarative (or static) knowledge (Card, Moran, & Newell, 1980, 1983). Hence the cognitive approach can identify the common elements “supplied by the behaving person as well as by the situation” (Garrett, 1941, p.96). This emphasis on common elements puts the cognitive science approach squarely in the tradition of Thorndike’s approach to transfer, the difference being that cognitive elements are much more abstract than the usual interpretation of Thorndike’s dictum as S–R pairs.

While the distinction between procedural and declarative knowledge serves to bring general transfer under the sway of a cognitive common elements approach, it also represents a major difference in the types of knowledge studied by the verbal learners and the cognitive scientists. Verbal learning studies of specific transfer (Postman, 1971) emphasized the acquisition of the declarative knowledge involved in learning lists of words or nonsense syllables. In contrast, the cognitive science studies that we review show an overwhelming tendency to study the transfer of procedural knowledge, specifically that knowledge involved in goal directed cognitive skills (for example, text editing, operating a device, solving the eight-square puzzle, and a performing a second-order tracking task).

B. THE SURPRISING SPECIFICITY OF TRANSFER

Commenting upon early research on transfer, Postman (1971) observed that “the repeated failures to find broad transfer effects implied that the habits permitting the efficient performance of a given task were highly specific and unlikely to generalize to new situations” (p. 1032). Such failure of transfer is found in recent studies as well. For example, Simon and Hayes (1976) found that people did not translate a difficult problem into one which was structurally isomorphic but much simpler. Likewise, Perfetto, Bransford, and Franks (1983) have found that people do not transfer declarative knowledge acquired in one condition to a new and seemingly relevant condition, despite evidence that the knowledge can be assessed under different eliciting conditions. Similarly, efforts to enhance intellectual performance by training cognitive skills have met with limited success. The skills simply do not transfer to novel contexts (Bransford, Arbitman-Smith, Stein, & Vye, 1985; Polson & Jeffries, 1985). Cognitive science approaches provide an insight into this surprising specificity of transfer.

The cognitive theorists, whose work we discuss, have focused on two areas of importance to transfer: The problem space per se and the procedures used to solve problems (or move) in that space. Transfer of cognitive skills involves either the use of a familiar problem space to solve a
novel problem or the use of well-learned procedures in new applications.

Problem space approaches to transfer emphasize how knowledge is used rather than what knowledge is used. Declarative knowledge is viewed as quite flexible, with the same bit of declarative knowledge used in many different ways. However, the various procedures in which the same bit of declarative knowledge is embedded can be quite different. Transfer is predicted when the same knowledge is used in the same way across different tasks. No transfer is predicted when the same knowledge is used in different ways (Anderson, 1987).

This emphasis on knowledge use is one of the major differences between cognitive science and older identical elements theory. The cognitive science approach does not view transfer as guaranteed merely because the same knowledge is present in A as is required to perform B. Not only must the knowledge be the same, but how it is used must be the same. The problem space hypothesis provides a way of representing knowledge that by making these distinctions clear provides insight into the surprising specificity of transfer.

**C. THE PROBLEM SPACE HYPOTHESIS: A BRIEF INTRODUCTION**

The problem space hypothesis (Newell, 1980) states that "the fundamental organizational unit of all human goal-oriented symbolic activity is the problem space" (p. 696). The constraints imposed upon a cognitive architecture by the problem space hypothesis can be easily mimicked by production system models. To understand the problem space hypothesis is to understand why many cognitive scientists use production systems as a tool for theory development as well as a convenient format for representation.

A problem space is a symbolic structure consisting of *states* and *operators*. Each operator takes a state as input and produces a state as output. A *path* is a sequence of operators that thread their way through a sequence of states. A problem in a problem space consists of a set of *initial* states, a set of *goal* states, and a set of *path constraints*. Solving a problem involves finding a path through the space, starting at an initial state, passing only along paths that satisfy the path constraints, and ending at a goal state.

As an example, consider the problem of trying to get from Washington, D.C., to San Francisco. The initial state is the Washington National Airport. Your goal state is the San Francisco Airport. The *states* consist of all airports in the United States. You have three operators: The *direct-flight* operator enables you to ask the ticket agent for a direct flight to some goal city, in this case San Francisco; the *next-flight-west* operator enables you to get on a plane heading west from where you are now; finally, the *next-flight* operator simply books you on the next flight out, regardless of destination. You may apply one of these operators to get to a new state (literally and
figuratively). For example, your next state could be airports in Philadelphia, Newark, New York City, Pittsburgh, Boston, Atlanta, Chicago, and so on. (Note that there are no direct flights from Washington National to San Francisco. This is an example of a path constraint.) Once in the next state, you can again apply an operator to reach another state. This process continues until you find yourself in the goal state, that is, the San Francisco Airport.

By the problem space hypothesis, problem solving is synonymous with searching the space. The search (that is, solving the problem) begins by applying operators to the current state. Each application of operator to state produces a new state, which is added to the stock of states. The new state is evaluated to determine if it is any closer to the goal state than the existing stock of states. If so, then an operator is applied to this new state; if not then another operator is applied to an old state. To carry out this search process, cognitive scientists postulate a fixed cognitive architecture or mechanism that works for all problem spaces.

Since the problem space hypothesis is a model of human goal-oriented cognition, human resource and capacity limitations impose constraints upon what can take place. The number of previous states which a human can remember affects the probability of revisiting a given state or of learning experience-based, short-cut methods. Speed of retrieval from long-term memory, number of chunks, and information per chunk, all affect the mental representation of the problem space and operations in the problem space.

The cognitive skills involved in activities as diverse as text editing, taking airline travel reservations, playing bridge, installing an electric light fixture, troubleshooting a radar system, programming, and so on, all can be said to occur in some problem space, and all are potentially representable within a production system framework. The strong claim is made that anything found to be true about the transfer of one cognitive skill can be applied to all other cognitive skills. Any exceptions must be capable of being explicitly accounted for within the problem space framework.

**D. TOOLS FOR PROBLEM SPACE THEORISTS: PRODUCTION SYSTEMS**

The production systems are special-purpose programming languages that seem especially suited to dealing with information-processing theories of human performance (Neeches, Langley, & Klahr, 1987; Newell & Simon, 1972). Production systems allow the full complexity of the human information-processing system to be represented and demand that theoreticians explain how the various mechanisms interact. Complexities such as under what circumstances processing involves short-term versus long-term
memory, parallel versus serial processing, or pattern matching versus problem solving must be considered and incorporated into the resulting theory.

Production system models are evaluated by criteria of sufficiency and power. A model is sufficient if its mechanisms and their interactions mimic human performance in some context. It is powerful if this mimicking extends over a wide range of inputs and contexts, such as programming in LISP, solving geometry problems, or text editing.

Given that production systems may be useful, what exactly are they? At the simplest level of description, a production system consists of two data structures which interact via a simple processing cycle. When modeling human cognition, the two data structures are thought of as working memory and production memory. The processing cycle consists of a three-stage recognize-act cycle. It matches one or more production rules (from production memory) with the contents of working memory, decides which of the matched rules to fire (sometimes called conflict resolution), and fires (the execution or act stage) the selected rule(s). Firing a rule may result in actions in the external world as well as changes to the contents of working memory.

The production rule given below (from, Neches et al., 1987, p.5) is taken from a set of production rules for subtracting two numbers:

```
FIND DIFFERENCE

IF: you are processing column,
    and number1 is in column and row1,
    and number2 is in column and row2,
    and row1 is above row2,
    and number1 is greater than or equal to number2,

THEN: compute the difference of number1 and number2,
    and write the result in column.
```

As with all production rules, this rule consists of a condition-action pair which will fire or execute based upon the outcome of the three-stage recognize-act cycle. In the match stage, the condition (or IF) side of all rules in production memory is compared with the contents of working memory (this is usually assumed to be a parallel process). In the production system from which the above rule is taken, the words in italics are considered variables. So, not only does the entire subcondition "number2 is in column and row2" have to match, but each variable must be instantiated with an element from working memory. (Note that not all production rules will have variables.)

After the match stage, the conflict resolution stage selects one or more production rules to fire or apply. How this selection process works depends
upon the theory being modeled. For a simple expert system that does not purport to model human cognition, this stage may be eliminated so that all productions that match are fired. Depending upon the theory, the conflict resolution stage may select those productions which match the most recent, the oldest, the strongest, the most important, and so on, information in working memory.

The act stage then executes the production rule(s) selected. Executing our sample production rule modifies the contents of working memory (by adding the result of the computation) and the external environment (by writing the result in the correct column).

Within the production system framework, different parts of this recognize-act cycle may be considered as involving serial or parallel processing. Likewise, different capacity, decay rate, and so on can be assumed for working memory as well as production memory. Since the firing of a production rule results in the modification of working memory, this recognize-act cycle can continue indefinitely. In practice, it continues until a certain prespecified goal state (such as solving a subtraction problem) is achieved.

E. PERFORMING IN A PROBLEM SPACE: PREREQUISITES TO LEARNING AND TRANSFER

Problem space theories of learning and transfer are firmly rooted in problem space analyses and theories of performance. To understand the problem space approach to transfer, the reader must have some acquaintance with these analyses. To this end, we introduce GOMS (Card et al., 1980, 1983).

GOMS stands for goals, operators, methods, and selection rules. The goals are the goals and subgoals of goal-oriented cognition. Beyond this, "the dynamic function of a goal is to provide a memory point to which the system can return on failure or error and from which information can be obtained about what is desired, what methods are available, and what has been already tried" (Card et al., 1980, p. 38). Operators are the elementary motor or information-processing acts that a person uses in performing the task. The operators that are selected for study depend upon the level or grain of analysis desired. As our interests go from the macro to the micro, the elementary operators we choose for our analysis go from a mixture of basic information-processing mechanisms and learned behavior to purely low-level information-processing mechanisms. For example, at a macro level, the elementary operators used to describe the process of preparing this chapter include writing, reading, talking, and listening. In contrast, a micro analysis of proofreading the chapter might examine more elementary memory retrieval and pattern-matching operators.
A method is a particular combination of subgoals and operators that can be used to accomplish a goal. There may be several methods available to accomplish any given goal. Which method is chosen depends upon the selection rules which provide the control structure of cognition. However, the selection of a method cannot be the result of an extended deliberation (Newell, 1980). If so, it would not be a selection rule but would itself be a method that is executed in another problem space.

GOMS is based upon general human information-processing theory (see especially, Card et al., 1983). While it was developed in the context of analyzing text-editing expertise (Card et al., 1980, 1983), it is not specific either to experts or to text editing. For example, it has been extended to analyze the learning (Kay & Black, 1985) and transfer (Singley & Anderson, 1985, in press) of text-editing skills and has begun to be used in other domains (Kieras & Polson, 1985).

F. SUMMARY AND OVERVIEW OF THE CHAPTER

We began this section by comparing the cognitive science and verbal learning approaches to transfer. Interestingly, both traditions base their transfer theories upon Thorndike's (1914) common elements theory. The verbal learning theories emphasized specific transfer and all but ignored general transfer. From the cognitive science perspective, this lopsided development stemmed from impoverished conceptions of learning and a complete absence of theories of performance. The theories of learning used by the verbal learners could not yield an analytic representation of the cognitive skills involved in such general transfer phenomena as modes of attack or the use of general principles. In contrast, these general transfer phenomena are exactly those aspects of transfer that the cognitive scientists have chosen to emphasize. The relative lack of attention given to specific transfer phenomena does not show some inherent limitation of the cognitive approach. Rather, it reflects a tradition which emphasizes skilled performance over rote learning. Despite these differences, we believe that there is much that the cognitive theorists can learn from the older verbal learning studies (especially in the area of methodology and measurement). Furthermore, the cognitive approach cannot be fully accepted until it provide an account of the specific transfer phenomena favored by the verbal learners.

The problem space is the fundamental architectural assumption of those cognitive scientists whose theories we review in this chapter. In the next section we discuss cognitive theories of between-task transfer. This is followed by a discussion of several theories of learning for which production system models have been created. We discuss research that shows how the ability to represent the interaction of knowledge and learning mechanisms in precise
ways explains and predicts within-task transfer. Finally, we summarize our conclusions about the utility of this new approach to transfer and where we see it heading.

II. TRANSFER THEMES FROM PROBLEM SPACE APPROACHES

Acquiring cognitive skill on a task involves acquiring knowledge of the states in the task problem space and knowledge of operators for moving from one state to another. Transfer of cognitive skills involves the transfer of knowledge about states and operators to new problems in the same or different problem space. Knowing what, at least, some of the states are in a new problem space is a tremendous aid to solving problems in that space. For example, knowing that a state $x$ exists which is closer to the goal state than is the current state (and is therefore a desirable intermediate step) allows the problem solver to set a subgoal of achieving state $x$. Likewise, having operators already organized into methods or procedures for moving between states provides a powerful set of ready-made tools.

Transfer of states and procedures (operators) that occurs totally within a given problem space is usually considered a type of learning and, as such, will be considered in the next section. In this section we consider transfer between problem spaces of varying degrees of dissimilarity.

Two sets of studies from two different laboratories are discussed. The approaches these laboratories take is as remarkable for the similarities as for the differences. Similarities include a strong commitment to the problem space hypothesis and a primary (but not exclusive) use of text-editing skills as the domain of transfer. Differences stem from the emphasis of one laboratory, Moran and associates, on the cognitive engineering of software/hardware design and training, and the other, Anderson et al., on extending a powerful theory of learning, ACT*, to account for transfer. A secondary difference in the two approaches is the emphasis of Anderson et al. on casting their theoretical mechanisms in a production system format.

A. A COGNITIVE ENGINEERING APPROACH TO TRANSFER

Learning a new cognitive skill or a new version of an old skill can entail borrowing pieces from somewhat similar problem spaces. Clearly there are cases where this strategy is advantageous (if not 100% positive), as when learning to drive a car with a stick shift after already knowing how to drive an automatic. However, it is interesting to ask whether borrowing a problem space may entail negative as well as positive transfer and, if so, to predict which type of transfer occurs when.
1. Borrowing a Problem Space: Analogical Models

Borrowing from other, already mastered problem spaces may involve the use of analogical models (Douglas & Moran, 1983; Halasz & Moran, 1982). Such borrowing entails mapping the known structure of one complex task onto parts of the new task. By this process, objects, relations, operations, and so on in the known source problem space are mapped, or transferred, to corresponding objects, relations, operations, and so on in the unknown target system (Douglas & Moran, 1983). For example, an analogical model might involve using what people know about file cabinets to teach someone how a computer file system works.

While analogy works fine for simple situations, it breaks down when elaborated to deal with the complexities of a new system. Continuing the example, how would you add the notion of password protection to the filing cabinet model of computer file systems? With such elaborations, the once simple analogical model quickly becomes “a baroque collection of special-purpose models pasted together in a more-or-less integrated and a more-or-less consistent fashion” (Halasz & Moran, 1982, p. 384).

A reliance upon analogical models produces two sorts of problems. First, the user has to decide which model out of his or her “baroque collection of special-purpose models” applies to the situation at hand. Second, an analogical model tends to explain too much. In his file cabinet, one of us has Halasz and Moran, Douglas and Moran, and Moran all filed in his “Moran” folder. Trying to save his notes on each of these papers in a computer file {DSK}<LISPFILES>NOTE CARDS>MORAN would have the effect of generating three different versions of the “MORAN” file. The file cabinet analogy, in this case, would mislead the user and interfere with the acquisition of a more accurate model of how a computer file system works. Problems with borrowing from analogical models lead to the conclusion that the best way to teach a novice to reason about a new domain is to create an abstract conceptual model that is tailored for the task at hand (Halasz & Moran, 1982). (This is similar to the conclusion that Piroli and Anderson, 1985, [see below] reach regarding what type of recursion example to teach novice LISP programmers.)

A problem with this conclusion is that few specialized frameworks exist that a novice can use in learning about a new system. Hence, while the use of analogical models may yield negative transfer when compared to some optimal tailor-made model, our intuitions are that analogical models yield positive transfer when compared to using no model (or building a brand-new model from scratch).

Douglas and Moran (1983) developed an analysis technique for generating a taxonomy of analogical misconceptions. The analysis focuses on operators in the GOMS sense. It looks at operators in the two problem
spaces (that of the analogical model and that of the to-be-learned model) and marks an operator in Problem Space 1 as similar to an operator in Problem Space 2 if the two operators produce similar effects (postconditions) or have surface features in common. Misconceptions arise when either the preconditions or the postconditions for use of the operator in the new problem space are not yet known. They result in similar operators from the analogical model being substituted for the unknown (or less known) operator from the new problem space. An example from the problem spaces of typewriting and text editing is the use of the space-bar operator. In typewriting, the space bar moves the print element to the next cell or character to the right. For text editing, the space-bar inserts a blank space between the current position and next cell or character to the right. This mix-up is a common mistake for novice text editors.

In analyzing the text editing and typewriter problem spaces, Douglas and Moran distinguish between locative and mutative operators. Locative operators move the pointer around (such as the space bar for typewriting and <control>-H for text editing) while mutative operators change the text. They show that three locative operators for the typewriter are similar to mutative text-editor operators which insert formating characters. In their observations of novice text editors, these three errors, involving an “ontological shift,” accounted for 30 out of 75 errors.

The operator-mapping technique shows the utility of the problem space construct for identifying points of negative transfer between two tasks. Douglas and Moran see their technique as a potentially useful design tool. We concur. For example, there are many times in private life, industry, and the military when people already familiar with one version of a piece of equipment, software program, and so on are required to deal with an “improved” version. Applying the operator-mapping technique to these new versions could result in engineering changes or, at least, identification of training needs.

2. Differences between Problem Spaces as a Measure of Transfer

Moran (1983) builds upon his interest in mapping operators between problem spaces to create an analysis framework for use in systems design and as a competence (performance) model of the user. This ETIT analysis consists of (a) an external task space, (b) an internal task space, and (c) a mapping from the external task space to the internal task space. The notion is that a user comes to a system with a task already formulated in terms external to the system. An example of a mapping problem is putting a household budget onto a computer spreadsheet. The budget-keeping task is already well formulated according to the user’s (probably idiosyncratic) bookkeeping methods. The problem is to reformulate the task to fit the spreadsheet. The system’s complexity relative to its target external task space is assessed
by examining its mapping rules. The technique can, at least in principle, be used to predict transfer among devices merely by gauging the differences between the mappings for different devices. The degree of overlap predicts the degree of transfer.

Moran has used ETIT to compare a line editor (LINED) with a display editor (DISPED). First, a mapping was done from the external problem space to each editor. These two mappings were then mapped onto each other. Every DISPED rule had a corresponding LINED rule. This suggests easy transfer from LINED to DISPED, but the transfer should not be as effective the other way. There were five LINED rules that had no correspondence in DISPED, and the LINED rules were more highly conditionalized than DISPED.

At this point ETIT is not a finished tool but is a research program based upon the notion that all goal-directed cognitive processes occur in a problem space of one sort or another. With further development, ETIT will be a useful tool for predicting ease of learning and operating new devices. Note that ETIT defines ease of learning and operating in terms of the mapping or number of common elements between how a person currently thinks about a task and the way that task must be performed if a given device is used.

3. Summary

Borrowing pieces of a problem space can be risky but beneficial. Carefully defining an analogy’s boundary conditions should help to maximize its benefits, although, when possible, a carefully crafted specialized framework is preferred.

A goal for cognitive engineering is to develop a metric for determining what aspects of one problem space transfer positively or negatively to another problem space. As witnessed by Douglas and Moran (1983) and Moran (1983), useful steps in this direction are being made. (Kieras and Polson’s 1985 model is similar in spirit to Moran’s; however, due to space limitations we chose to discuss the Moran model as illustrative of this type of cognitive engineering effort.)

B. PRODUCTION RULES AS THE ELEMENTS OF TRANSFER: ACT* AND ITS IMPLICATIONS

A prerequisite to the common elements approach to transfer is being able to precisely specify what the common elements are. We believe that it was the ability to describe common elements in terms of S and R features and the corresponding inability to precisely describe cognitive elements that drove the verbal learners to emphasize specific transfer and to all but ignore general transfer.
The problem space hypothesis provides us with a basic vocabulary to describe the states and operators involved in performing a cognitive skill. Likewise, GOMS (Card et al., 1980; 1983) provides a way to increase our analytic precision by analyzing expertise as operators organized into methods to accomplish certain goals when selected according to certain rules. Yet another step in precisely describing the cognitive elements is provided by the production rule.

Anderson (1987) takes a strong view of production rules as the common elements of transfer for which Thorndike was searching. Of course, production rules are more abstract than the usual interpretation of Thorndike’s dictum as S-R pairs. For ACT* (Anderson, 1983, 1987), this abstraction comes in several ways. First, a production rule with variables (see section I.D.) in its conditions functions as a template which can be matched to a family of patterns rather than one specific pattern. Second, the function of many production rules is to generate hierarchical goal structures (to control behavior) rather than to execute a specific behavior. Third, the conditions and actions of many production rules deal with elements of cognition and are likely to affect change in the internal, cognitive environment as in the external one.

Singley and Anderson (in press) propose an explanation of the specificity of transfer that is essentially an extension of Anderson’s theory of learning (Anderson, 1982, 1983, 1986, 1987). By their view, the elements of transfer are subsets of the elements of learning, which makes any common elements approach to transfer totally accounted for by learning theory.

We begin this section with a discussion of ACT*'s learning mechanisms. While it is beyond the scope of this chapter to give a detailed exposition of ACT*, we intend to provide just enough information about the theory so that the reader can appreciate ACT*'s account of the specificity of transfer: It is not what, but how, knowledge is used that is important.

1. A Selective Overview of ACT*

"Knowledge comes in a declarative form, and is used by weak methods to generate solutions, and the knowledge compilation process forms new productions. The key step is the knowledge compilation process, which produces the domain-specific skill" (Anderson, 1987, p. 197). This statement is central to ACT*'s account of transfer. When learning a new domain, general, nonspecific problem-solving procedures (production rules in ACT*'s implementation) are used to interpret declarative knowledge in working memory. Skill acquisition proceeds through the processes of knowledge compilation that gradually build up a set of domain-specific
productions. These new productions speed performance by the processes of *composition* and *proceduralization*. The composition process speeds performance by collapsing general procedures into a smaller number of more specific procedures. Proceduralization eliminates the need to retrieve repeatedly the same knowledge from declarative memory by directly incorporating that knowledge into new productions.

For ACT*, cognitive skill acquisition is a goal-directed, problem-solving activity. Hierarchical goal structures underlie all of ACT*'s activities. Existing hierarchical goal structures direct performance, learning, and transfer and ensure that any new productions are embedded with new goal structures. These goal structures account "for the ubiquitous hierarchical structure of human behavior, and the focus of attention it produces explains the strong seriality in the overall flow of human cognition" (Anderson, 1983, p. 33). Thus, ACT* can be described as a theory of learning by contiguity, where contiguity is not temporal contiguity, but contiguity in the goal structure of problem solving.

2. Learning by Doing and Other Implications of Knowledge Compilation

The ACT* distinction between declarative and procedural knowledge, how the two are used, and how they are acquired has several implications for transfer.

Declarative knowledge is flexible and not committed to how it will be used. However, knowledge compilation often derives productions from declarative knowledge that can only be used in certain ways. Often the production sets underlying different uses of the same knowledge can be quite different. (Anderson, 1987, p. 201)

Declarative knowledge is accessible to a large number of general purpose (or weak) problem-solving methods. These methods act upon declarative knowledge in an interpretative mode, which produces performance that is slow and subject to errors. Since all behavior (cognitive and physical) is under the control of procedural memory, skilled performance requires that these slow, interpretative procedures be transformed into domain-specific procedures. This transformation occurs through the knowledge compilation mechanisms of composition and proceduralization.

The way knowledge compilation is carried out has several consequences for learning and transfer. For composition to occur, productions with *contiguous goal structures* must be present together in working memory. Since these productions are specific to skilled performance, the best way to ensure that they will occur together in working memory is by practicing the skill itself. In a very real sense, the ACT* system can only learn skills by doing
them. The unequivocal prediction is that for cognitive skill acquisition, practice is more important than formal instruction.

The evidence supporting explicit instruction in domain-specific problem-solving strategies is fairly strong. In a recent review of the literature on troubleshooting, Morris and Rouse (1985) conclude that "either troubleshooters should be explicitly instructed in how to approach problems or they should be forced to use their knowledge of the system explicitly in deciding what to do" (p. 527). In ACT* terms, the former provides novice troubleshooters with declarative representations of the appropriate procedures, while the latter forces them to apply general problem-solving methods to their declarative knowledge. In either case, the result is that troubleshooting procedures with contiguous goal structures are likely to be in working memory at the same time and, therefore, available to the knowledge compilation process.

This description of procedural learning has a direct consequence for transfer. Namely, there is no reason to predict transfer between different uses of the same knowledge. Practicing troubleshooting, for example, may strengthen the declarative representation of troubleshooting knowledge while that knowledge is being interpreted by general problem-solving productions; however, this stage passes quickly. Once the knowledge compilation mechanisms start forming new domain-specific productions, then further practice results in increasingly specialized productions. To the extent that the transfer task requires the same procedures, positive transfer will occur. To the extent that it requires different procedures (even those based upon the same declarative knowledge), transfer will be zero. Several studies support this prediction.

Neves and Anderson (1981) found that 10 days of practice giving reasons to justify steps of a worked-out geometry proof had little effect on students' ability to generate a proof. Although the declarative knowledge for the two tasks is the same, there is no overlap in the production rules for generation versus justification of steps.

A similar result was found by McKendree and Anderson (1987) in a study that was concerned primarily with studying knowledge compilation processes. They gave 20 subjects 4 consecutive days of practice in evaluating combinations of four basic LISP functions. Two functions combined items into a list (INSERT and LIST), the other two extracted items from a list (CAR and CDR, but assigned the more mnemonic names of FIRST and REST). Subjects had 150 trials each day for 4 days, and performance improved following a power law function (Newell & Rosenbloom, 1981) in a manner predicted by ACT*.

While these results are interesting, we are more concerned with the transfer data that McKendree and Anderson collected more or less as an aside to the main experiment. After the first and last session, subjects were
given a transfer task that required them to generate functions similar to those they had just practiced evaluating. All of the transfer problems involved basic functions or pairs of functions that the subjects had seen immediately before in the evaluation task. Yet, while error rates on the evaluation task decreased dramatically from Day 1 to Day 4 (35% to 15%) the transfer task showed little improvement (29.3% versus 26.6%). Despite becoming significantly better at evaluating functions, subjects are not any better at generating functions than they were on Day 1.

C. TRANSFER OF PRODUCTIONS AMONG TEXT EDITORS: SINGLEY AND ANDERSON

Powerful techniques are required to build theories that predict when and how much transfer will occur. Indeed, theoretical progress on the transfer of cognitive skills has been limited by the absence of techniques precise enough to build a sufficiently detailed empirical data base. The studies reported by Singley and Anderson (1985, in press) are an important step in remedying this deficit. They apply the analytic power of ACT* (Anderson, 1983) and GOMS (Card et al., 1980, 1983) to keystroke data to produce powerful analysis techniques which are applied to study the complex cognitive skill of text editing.

(While the military and industrial literatures contain many studies of the transfer of complex skills, as far as we know, all of these have focused on relatively large-scale measures of transfer. The empirical data base that Singley and Anderson [in press] desire would be based upon the detailed analysis of the cognitive skills required to perform the task [performance model]. The analysis of transfer would examine how the various goals and subgoals, operators, methods, and selection rules transfer from one task to another.)

The domain of text editing was picked because of the existence of independently performed task analyses of the productions involved. The production system-like models that existed (Card et al., 1983; Kieras & Polson, 1985) served to constrain the ACT* production system models (Anderson, 1987). These exploratory studies were to observe and characterize the magnitude and direction of transfer between different text editors, the components of learning and transfer, the grain size that would yield the most interesting data, and any strategic components that were important, such as goal structure.

1. Methods and Macroanalyses

In the experiment reported in both the Singley and Anderson papers (1985, in press), transfer among two-line text editors (UNIX ED, VMS EDT) and a screen editor (UNIX EMACS) was studied. For each editor a
subset of commands was chosen. Expert typists, with no prior text-editing experience, edited 3 h a day for 6 days. All groups spent the last 2 days using EMACS. The ED and EDT group spent the first 4 days on ED or EDT respectively. The ED-EDT and EDT-ED groups spent 2 days on one line editor and the next 2 days on the other. The EMACS group spent all 6 days on EMACS. The control group spent 4 days typing at the terminal. Keystroke data were recorded.

Transfer data from the macroanalyses showed near total transfer between the two line editors (Days 3 and 4), moderate transfer from line editors to screen editor (Days 5 and 6), and slight transfer from typing-only to screen editor (Days 5 and 6). All measures of macrotransfer (including reduction in total time, reduction in total keystrokes, reduction in residual errors, and increase in keying rate) concur with these overall statements.

2. GOMS-based Microanalyses

The most interesting data are provided by microanalyses that vary the grain size of transfer. These GOMS-based (Card et al., 1980, 1983) analyses consider text editing as a series of largely independent unit tasks. (A unit task is a text-editing operation. Delete character, insert word, replace line are three examples of unit tasks.) Three subgoals are required to accomplish each unit task: encode the edit from the manuscript (acquire-unit-task), move to the line requiring modification (locate-line), and modify the text (modify-text). This analysis provides the hierarchical goal structure shown in Figure 1.

Singley and Anderson (in press) developed a parsing algorithm to identify each burst and pause in the keystroke data and attribute it to the planning or execution of each unit task as well as its major subgoals: acquire-unit-task, locate-line, and modify-text. First, for each page, the algorithm segments the data into six unit-task episodes, one for each edit on the page. Second, it subdivides each segment into a locate-line (LL) or modify-text (MT) component. The LL component includes time and keystrokes spent moving to the site of the modification and time spent acquiring the unit task from the manuscript. The MT component includes the time and keystrokes needed to modify the text. Third, each component, LL or MT, is split into planning and execution subcomponents which are defined as follows: Execution equals the time from first to last keystroke minus any inter-keystroke pauses of greater than 2 s. Planning equals the sum of all pauses greater than 2 s.

To summarize the microanalyses, the parsing algorithm yielded learning and transfer curves for each of the 12 text-editing operations (unit tasks) that subjects performed. Each curve was further divided into LL and MT components and divided again into planning and execution.

The learning curves for each kind of modification (unit task) had a
characteristic shape regardless of editor. For simplicity, Singley and Anderson (in press) discuss the curve yielded by the delete-character unit task. The total time for this edit dropped from 74 s/edit on Day 1 to 25 s/edit on Day 4. However, a finer grain analysis showed that execution time for this edit was constant with all the improvements in seconds per edit due to reductions in planning time.

Next, Singley and Anderson report the results of examining six dependent variables ([planning time, execution time, and number of keystrokes] × [LL and MT]) for each editor. The interesting finding is that EMACS is not uniformly superior to the line editors on all components. Its advantage is primarily due to more efficient MT methods that require fewer keystrokes and less planning time. For the LL component, ED is superior to EMACS (difference in execution time is significant; difference in planning time is not).

Based upon their task analysis (see Figure 1), Singley and Anderson (in press) predict differences in the pattern of transfer among the three editors for the LL versus MT component. For LL all three editors use different methods to locate lines; but, the LL component spans not only locate-line procedures but also acquire-unit-task procedures and the superordinate goal nodes (edit-unit-task and edit-manuscript). Therefore, “traversing the top nodes in the goal tree and encoding the edits are the same regardless of editor” (p. 34); they predict moderate and equal degrees of transfer among the three editors for the LL component.
For MT, the predicted pattern is different. For the line editors the surface features of the MT commands differ, but the underlying conceptual structures are nearly identical. In contrast, for the line editors (ED and EDT) versus the screen editor (EMACS), the MT procedures are completely different. For the MT component they predict nearly total transfer between the line editors but essentially none from the line editors to EMACS.

For line editors, as predicted, they find more transfer for the MT component (102%) than for LL (89%). (Note that by the formula used, greater than 100% transfer occurs when the transfer groups [ED-EDT and EDT-ED] do better on Days 3 and 4 than the groups that do not switch [ED and EDT only].) However, they predicted moderate transfer for the LL component while transfer, in fact, was high. The keystroke data suggests that the LL component is inflated due to the increased use among transfer subjects of a common secondary method for addressing line location, namely, the use of carriage returns rather than line addressing (for example, \texttt{J0p}) in ED and string searching (for example, \texttt{t \textasciitilde unique}) in EDT.

Transfer from the line editors to EMACS yielded surprising findings. Singley and Anderson found 61% transfer of the LL component from the line editors (LineEd) to EMACS and 35% from the typing control (TypeCon) group (average of 48%). For the MT component, transfer was 62% (LineEd) and 29% (TypeCon) (average of 46%). Hence, contrary to prediction, the LL component is not as large for transfer from line editors to EMACS as from one line editor to another. Also, transfer of the MT component is as large as for the LL component.

While Singley and Anderson (in press) report many of these detailed and interesting analyses, we will focus on one last result: the attempt to determine why there was so much MT component transfer to EMACS from both the line editor (LineEd 62%) and typing control (TypeCon 29%) groups. Candidate hypotheses include: portions of the upper-level goal tree (LineEd only) and subskills for encoding the edit from the manuscript (LineEd and TypeCon).

Singley and Anderson (in press) first look at all 12 editing operations and find no evidence for any general trend. They then decide to focus on those subgoals that show the most and the least transfer to EMACS. Delete-Character shows the most transfer for both groups (LineEd 90%, TypeCon 89%), Delete-String the least (LineEd 28%, TypeCon –98%). Delete-Word shows strong transfer for both groups as well (83% and 74%). DeleteLine shows strong transfer only for LineEd groups (77% and 29%).

Both the LineEd and TypeCon groups learned a deletion operation that can transfer to EMACS: the delete key. This explains why delete char and word is so high whereas delete string is low (there are much easier ways for EMACS to delete a string than using the delete key). This hypothesis predicts that the transfer subjects should use the delete key more often than
the EMACS only subjects. Indeed, the transfer groups used the delete key 68% of the time versus 50% for EMACS only. This difference was not significant but is in the right direction.

The delete-line procedure has same goal structure in EMACS as in the line editors, but not for TypeCon. For both EMACS and LineEd the user moves to the line requiring deletion and types the delete line operator (typing .d in ED and d in EDT, and <control>k in EMACS).

3. Summary of Singley and Anderson

The Singley and Anderson experiment represents a milestone in the examination of the components of transfer. Using a problem space–based analysis, GOMS, they took apart a complex cognitive skill, text editing, and were able to examine the various components to determine the locus of transfer effects.

As they point out, the number of methods for performing any one edit makes quantitative predictions very difficult. For example, the LineEd and TypeCon groups were able to “fixate” on using the delete key rather than learning more efficient EMACS-specific methods for deleting characters and words. Analyses at a bigger grain size completely missed this phenomenon.

A next step in understanding the transfer between text editors would be to derive an ACT*–based production system simulation. Such an analysis would demonstrate the power of the ACT* theory to account for detailed transfer data. This extremely time-consuming endeavor would be justified by the increased understanding in the transfer of complex cognitive skills.

III. LEARNING-AS-TRANSFER

It is not possible at present to formulate a theory of transfer which will incorporate all of the [known facts of transfer] in a thoroughly systematic manner. . . . When such a system is constructed, it will take into account . . . the necessary transfer from trial to trial during practice. . . . and the other ways in which transfer pervades a majority of the phenomena of learning. (McGeech, 1942, p. 438)

The issue of learning-as-transfer is important in structuring any program of instruction that teaches a complex skill. For example, programming or operating a device involve skills that are so complex that we do not expect students to master them at one sitting. In fact, for many such skills we do not expect to teach students all they need to know. Rather, the best we can do is to teach them a critical subset of skills that will enable them to tackle and solve new problems as they arise. Unfortunately, we still do not have a learning theory that will enable us to design an optimal program of instruction. We are, however, beginning to get answers to the question of what is learned in one exercise (or session) that can be transferred to the next.
In this section we discuss four different approaches to learning within a problem space context. For each we will sketch its theoretical assumptions and present some empirical evidence that addresses the relation between learning and transfer.

A. BUILDING A PROBLEM SPACE: MENTAL MODELS

Section I.C. emphasized a novice's search in a problem space in which only the initial state, goal state, and set of operators was known. In a reasonably complex problem space, such as for troubleshooting a radar system, many problems each with a different initial state and a different goal state will be encountered. With much experience the problem solver (no longer a novice) acquires knowledge about many states in the problem space and how to reach a given state \( x \) from state \( y \) by the application of various sequences of operators. (In GOMS/ACT\(^*\) terminology, states along the path between \( x \) and \( y \) are treated as subgoals to be achieved by applying various combinations of operators. Alternative paths to state \( x \) are composed of different combinations of states and operators. These paths become compiled into methods, and so on). Depending upon the problem space chosen to represent the task, as well as other factors, at some point the problem solver becomes able to describe probable paths between two states for which a known path does not exist. At this point, the problem solver has a mental model of the problem space.

In our view, the difference between mental models and analogical models (discussed in section II.A.1) is one of emphasis, not essence. Research on both is concerned with defining the types of conceptual knowledge used to accomplish a task. Research on analogy tends to emphasize how conceptual models of Task \( a \) are used in learning or performing Task \( b \). Research on mental models tends to emphasize either the existence and importance of such conceptual knowledge in performing Task \( a \) (Gentner & Stevens, 1983) or how such knowledge is acquired when it does not already exist (Kieras & Bovair, 1984; White & Fredericksen, 1986). Both can be described in terms of the problem space hypothesis.

Research on the effectiveness of mental models for aiding learning and transfer is just beginning. Kieras and Bovair (1984) taught their subjects a Star Trek-like model of a "phasor-bank" device. The explanation was in terms of physical principles which do not exist, that carried out tasks which are nonexistent. However, subjects who were taught this mental model were able to learn procedures faster, retain them better, and execute them faster than subjects not taught the mental model. In a very different task, Eberts and Schneider (1985) compared performance of subjects on a second-order tracking task. Two groups were given augmentation cues while learning the task; a third group received no cues. For the augmentation groups, one
group was given a standard augmentation cue which Eberts and Schneider believed was not conducive to developing a mental model, while the other group was given a cue that was hypothesized to lead to the development of accurate mental models. In the absence of augmentation cues, the three groups did not differ in either performance or speed of learning (over 5 days). However, in a series of transfer tasks, the mental model group was consistently superior. (This manipulation is reminiscent of one used by Judd, 1908, with similar results. In that experiment two groups of boys practiced hitting targets underwater, first under 12 in. and then under 4 in. Prior to the first task, one group was taught the principle of refraction; the other was not. Performance on the first task did not differ; however, performance on the second task consistently favored the refraction group.)

Teaching a problem solver a mental model is a type of learning-as-transfer. Mental models contain declarative knowledge about the states and combinations of methods required to solve problems in the task problem space. After learning a mental model, the problem solver should be able to apply his, hers, or its general problem-solving strategies to the declarative representation of the mental model to determine what path through the problem space leads to the goal state. The problem-solving behavior of a novice who has been taught a mental model should have much in common with an expert’s behavior. (For example, while performance should be slower and subject to errors due to working memory failures, more expertlike forward, as opposed to backward, search may be shown.)

B. PRODUCTION SYSTEMS FOR COGNITIVE ENGINEERING: KIERAS AND POLSON

In a series of studies, Kieras and Polson modeled the procedures involved in operating different devices as sets of production rules (Kieras & Bovair, 1986; Kieras & Polson, 1985; Polson & Kieras, 1984, 1985). Each production rule was translated into an English sentence, and the entire procedure was taught to novices as declarative knowledge. Despite the declarative representation, they found that the number of new productions predicts how fast a new procedure will be learned, while the number of productions that two procedures have in common predicts the amount of positive transfer from one to the other.

The Kieras and Polson production system is the “most elementary form of production system in that there are no conflict resolution rules, and very simple kinds of pattern matching are used in evaluating conditions” (Polson & Kieras, 1985, p.207). Likewise, they make no attempt to represent fundamental cognitive processes such as memory retrieval or reading comprehension. This limitation greatly simplifies the production system without adversely affecting its utility for cognitive engineering of man-machine interfaces.
Their system assumes that each production takes the same amount of time to learn or execute. It predicts that the time to learn a new procedure depends upon the number of new productions in its description. From this prediction, a simple common elements theory of transfer is derived: The more procedures that two tasks have in common, the greater the transfer. (Note that this theory cannot predict negative transfer.)

1. The Experiments

Kieras and Polson have tested their production system in a series of three experiments (Kieras & Bovair, 1986; Polson & Kieras, 1984, 1985). The results are consistent across tasks (text editing, Polson & Kieras, 1985; operating a computer, Polson & Kieras, 1984; and operating a simple device, Kieras & Bovair, 1986) and across minor variations in procedures. For the sake of simplicity, we focus here on Kieras and Bovair (1986).

Kieras and Bovair (1986) required their subjects to learn 10 procedures for operating a device. For each procedure a GOMS-based analysis was performed and translated into a set of from 6 to 8 production rules. Instructions were prepared for each procedure by translating each production rule into a sentence. The 10 procedures were taught in one of three fixed training orders with substantial variation in the number of new rules in each procedure and in each serial position. (For example, assume that Procedure A involves learning 8 rules, of which 3 are in common with Procedure B and none are in common with procedures C or D. In order B A C, A has 5 new rules, 3 old rules (for 8 total). In order C D A B, A has 8 new rules.) For each procedure, total training time equals study time plus test time until the procedure was performed correctly three times in a row.

For each procedure, total training time varied as a function of what procedures (with what production rules) were learned before it. "The number of new rules alone accounts for 69% of the variance and is a better predictor of training time on a single procedure than the subjects' individual means" (p. 518). In addition, the reading times for instructional steps varied greatly depending upon whether the step was new or identical. This difference appeared on the first reading, implying that subjects immediately recognized new rules versus known rules and changed their study times accordingly.

In similar experiments, Polson and Kieras (1984, 1985) found that the number of new productions accounted for 61% and 83%, respectively, of the variance of learning time with approximately 30 s required to learn a new production (Polson & Kieras, 1985).

2. Summary: Kieras and Polson

These results show that production rules can provide a precise characterization of what is to be learned (Kieras & Polson, 1985). Such a production rule analysis could be used to derive an instructional sequence,
for example, that is uniformly challenging but never overwhelming to the student. However, the connection between Kiers and Polson’s use of production rules and that of other researchers is not as direct as it may at first seem. For most researchers, production rules, typically are used to describe well-learned skills. For Kiers and Bovair, 1986, the earliest stage of skill acquisition involves the translation of an “instruction sentence into a declarative representation of a production rule.” Next, this declarative representation is interpreted by a general procedure for following instructions and, eventually, the procedural form of these rules is created by processes such as those postulated by Anderson (1983; see section II.B.1. above).

Kiers and Polson’s research shows the utility that a relatively simple (from the psychological viewpoint) but powerful (because of the production system framework) model can have for cognitive engineering. The model is simple because it provides an essentially static representation of novice knowledge structures and does not represent fundamental cognitive processes. It assumes the operation of general procedures for operating upon declarative knowledge but postulates no mechanisms for acquiring greater levels of expertise. However, the seriousness of this limitation depends upon exactly how the model is used. Kiers and Polson are primarily interested in how such a simple model of novice memory structures interacts with a formalism they have developed (Kiers & Polson, 1985) to describe devices. Given this finely focused interest, their model seems adequate. In fact, we predict that the Kiers and Polson approach is the forerunner to a series of production rule models for cognitive engineering, each of limited scope, but each yielding useful prescriptions for real-world problems.

C. CHUNK THEORY MEETS THE PROBLEM SPACE: NEWELL AND ASSOCIATES

Newell and Rosenbloom (1981) set upon the task of trying to explain the “ubiquitous law of practice” and, in the process, postulate a powerful learning mechanism that has direct implications for within- and between-task transfer.

Skilled performance shows continual improvement with practice. This observation holds true for cigar making (3 million trials), choice-reaction-time tasks (40,000 trials), recall from long-term memory (approximately 5,000 trials), keystroke editing (approximately 1,000 trials), geometry proof justification (over 100 trials), the game of solitaire (500 trials), and many others: Newell and Rosenbloom considered many functions that might fit this phenomenon and concluded that the more general form of the power law is the best. That is, plotting trials against time on log-log paper yields a straight line that follows the general form of the power law:
\[ T = A + B(N + E)^a \]

where \( T \) = time
B = performance time of first trial
\( a \) = slope
A = the asymptote of learning as trial number \( N \) increases indefinitely
E = the number of trials of learning that occurred prior to the first trial measured (that is, prior experience).

Satisfied with the power law description, they went on to consider various ways in which such a function could arise and finally proposed the chunking hypothesis: "A human acquires and organizes knowledge of the environment by forming and storing expressions, called chunks, which are structured collections of the chunks existing at the time of learning" (p. 41). To this they add three assumptions:

1. Performance assumption: The performance program of the system is coded in terms of high-level chunks, with the time to process a chunk being less than the time to process its constituent chunks.
2. Task structure assumption: The probability of occurrence of an environmental pattern decreases as pattern size increases.
3. Learning assumption: Chunks are learned at a constant time rate on average from the relevant patterns of stimuli and responses that occur in the specific environments experienced. (p. 42)

With chunking as the only learning mechanism, Newell and associates develop two production system architectures, XAPS and Soar, which have interesting implications for the relation between learning and transfer. Of the two, we focus our discussion on Soar, which Newell and associates are developing as a general architecture of cognition.

1. Soar

Soar (Laird, Rosenbloom, & Newell, 1984) is a production system which treats chunking as a general learning mechanism and combines it with a general problem space problem solver. Chunking is a data-driven recorder of goal-based experience. The chunks formed represent the processing involved in achieving a subgoal in a problem-solving task. The next time the same subgoal is encountered (or generated by the problem-solving mechanism), the chunk is used in place of the rather complex problem-solving processes that were originally required.

Soar has "a reflective problem-solving architecture that has a uniform representation and can create goals to reason about any aspect of its problem-solving behavior" (p. 188). Implementing the relatively simple, but powerful chunking hypothesis within an architecture based upon the even more powerful problem space hypothesis expands the role of chunking.

1. Chunking can be applied to a general problem-solver to speed up its performance.
2. Chunking can improve all aspects of a problem-solver's behavior.
3. *Significant transfer of chunked knowledge is possible via the implicit generalization of chunks* [italics added].
4. Chunking can perform strategy acquisition, leading to qualitatively new behavior. (p. 188)

In Soar, both problem solving and routine cognitive processes are formulated as heuristic search in problem spaces. Soar can automatically generate subgoals for problems in any aspect of its problem-solving process. This ability is referred to as universal subgoaling and is an innovation of the Soar production system. Chunking acts on this subgoaling process by replacing the problem-solving activity with a production that can be used the next time the subgoal is encountered.

As chunking of search control knowledge continues, performance is improved by a continual reduction in the amount of search. Eventually, enough search control knowledge may be chunked so that no search is required, and what started out as a problem-solving activity ends up as an efficient algorithm for a task.

2. Transfer Implications of Soar

Based on the architecture of Soar and how chunking is implemented within it, several transfer issues emerge. (This is an excellent example of the constraints upon a transfer theory which emerge from a well-specified learning theory.)

If a given task shares subgoals with another task, a chunk learned for one task can apply to the other, yielding *across-task* transfer of learning. *Within-trial* transfer of learning can occur when a subgoal arises more than once during a single attempt on a task. Generality is possible because a chunk only contains conditions for the aspects that were accessed in the subgoal. This is an *implicit generalization*, by which many aspects of the context—the irrelevant ones—are automatically ignored by the chunk. (Laird *et al.*, 1984)

Laird *et al.* (1984) tested the implications of Soar with chunking in a series of simulations. They first evaluated the effect of the chunking construct in learning to solve the eight-puzzle problem. (The eight-puzzle is the familiar child's toy in which eight plastic squares with numbers on them are in a 3 × 3 space frame. The goal is to change the initial configuration of plastic squares into a goal configuration by moving one square at a time into the one empty square in the frame.) Because this is a computer simulation, Laird *et al.* were able to compare how long Soar took to solve the eight-puzzle problem with chunking (Soar-with) and without (Soar-without). Soar-without required 10 *evaluate-operator* subgoal searches. Soar-with required only 5. This dramatic reduction in search shows *within-trial transfer*.

They then gave Soar an *across-trial* transfer task. In this task Soar first solved a new eight-puzzle problem in which the initial and final states were different from the original and no intermediate states were the same. Chunking was turned on (soar-with) for this puzzle. After Soar had solved
this, chunking was turned off, and Soar was then given the original puzzle (soar-without). (Note that since Soar is a computer simulation, it was relatively easy for Laird et al. to induce complete amnesia for the original task.) Performance on the original problem in this across-trial task was much better than Soar’s performance in the above-reported within-trial task. Transfer occurred because in solving the original puzzle in the across-trial task, subgoals were generated that were identical in all of the relevant ways to subgoals in the new puzzle. As predicted, Soar showed implicit generalization when the only thing in common between the two puzzles were some of the subgoals created by Soar’s problem-solving processes.

In further simulations with tick-tack toe, Soar exhibited overgeneralization resulting in negative transfer. The next step in the Soar project is “to investigate how a problem solver can recover from over-general knowledge, and then carry out problem-solving activities so that new chunks can be learned that will override the over-general chunks” (p. 191).

Soar is an interesting example of the use of computer simulations in theory development. Only in a computer simulation could within- versus between-trial transfer be neatly turned on or off to examine the effect of one in isolation from the other. For theory development, this control over the theoretical mechanisms allows the theorists to study how much each postulated mechanism contributes and how it interacts with other mechanisms. Likewise, the overgeneralization shown by Soar in solving tick-tack-toe problems is an example of a simulation leading to the identification of a too powerful mechanism and the recognition that the theory has to be extended to explicitly model discrimination-type processes.

D. THE IMPORTANCE OF A GOOD EXAMPLE: ACT*

The ACT* theory predicts that what students learn from solving a problem, and how they try to solve subsequent problems, are very much influenced by the examples provided. For ACT*, powerful learning mechanisms “summarize solutions to these initial problems into new problem-solving operators which can apply to future problems” (Pirolli & Anderson, 1985, p. 242). As these new problem-solving operators are built, students rely less upon analogy and more upon these domain-specific operators. It is easy to see how the generality of the new problem-solving operators should be greatly influenced by the example used. This is the topic studied by Pirolli and Anderson (1985).

Using an interactive combination of protocol analysis and computer simulations, Pirolli and Anderson took protocols from students trying to solve recursion problems in LISP and then created a simulation of each student. The simulation was initialized with a set of productions assumed to represent the student’s prior knowledge of LISP. To this initial set was added the student’s solution to the first problem. The simulation was run, allowing ACT*’s knowledge compilation process to add new production
rules to the model. Note that after production rules simulating the student's solution of the first problem were added to the initial set of production rules, the only way additional productions were added was by ACT*'s knowledge compilation processes. Thus the results of the compilation process represent predictions of the way the next problem will be solved. Both ACT* and the student are then given the second problem and so on. ACT*'s model of knowledge compilation is validated to the extent that each student actually solves his or her second problem the way that ACT* does. The results of this study support ACT*'s predictions. In general, ACT* predicted the approach that each student used to attack each problem. Where the student had a difficulty solving a new recursion problem, so did ACT*.

Of more direct relevance to transfer is the effect that different types of examples had on the students' and ACT*'s success at solving later problems. First, Piorlli and Anderson note that the analogical mapping that students were performing was "never a mindless symbol-for-symbol mapping. Rather, it involved the subjects' knowledge of LISP and a representation of the meaning of what was mapped" (1985, p. 258). Second, the particular example provided did affect the way the students tried to solve later problems, as well as their success.

A contrast between two students, SS and AD, is particularly revealing. SS was provided with the recursion example used in a well-known LISP textbook. AD was provided with a description, or template, of recursive functions that was derived from the expert model. The textbook example was of a particular type of recursion. SS was able to solve recursive problems of this type but required help to solve different problems (such help was not available in the textbook). In contrast, AD's example provided her with a general strategy for coding a large class of recursive functions.

Piorlli and Anderson report an experiment which supports their empirical analysis. Two groups of subjects were provided with instruction on recursion and then required to write four training problems. The process group was given the standard textbook description (which emphasized how the process of recursion worked), while the structure group was provided with instruction that emphasized the structure of recursion problems. The structure group took 57.4 min to write correctly four training problems, while the process group took 85.3 min (a savings of 33%), but the groups did not differ in the time or correctness of writing 16 transfer problems. Piorlli and Anderson suggest that to deal with the four training problems, both groups had to acquire the same set of productions; however, the structure group "got to this state in a more efficient manner because they had learned a general strategy for structuring their code very early on in the training phase" (p. 271).

The Piorlli and Anderson study supports ACT*'s knowledge compilation process. By knowledge compilation, general or weak problem-solving
methods acting upon declarative knowledge are transformed into more powerful domain-specific methods. Furthermore, by knowing something about the student’s instructional history, ACT* is able to predict what domain-specific methods will be formed and which problems they will solve.

IV. SUMMARY

A fast summary of this chapter is that the common elements theory of transfer is alive and well but looks very different than it used to. What has been gained and what has been lost in the translation from verbal learning to problem space theories of transfer? At this point we believe that much has been gained and that the only losses are short term.

Among the most important gains is a concern with the procedural knowledge involved in performing complex cognitive tasks. While the approach is still too new to judge, there is significant progress in understanding how skills as complex as text editing, LISP programming, operating a device, and solving the eight-puzzle are transferred from one set of tasks to another. This gain should not be underestimated. The verbal learning approach never attempted an analysis of skills at such a level of complexity and, indeed, it is hard even to imagine how the verbal learning analyses could be applied.

Another gain is the recognition that common elements include such purely cognitive constructs as goals and subgoals, operators, methods, and selection rules. An example of the importance of knowing the goal structure of a domain can be attested to by anyone who has learned two text-editing programs. As Singley and Anderson (1985, in press) point out, the goal structure of different text-editing programs is largely identical, as are the individual goals and subgoals (for example, moving text, inserting text, altering text, and formatting paragraphs). These abstract goals and goal structures are readily transferred to any new text editor. Singley and Anderson (1985, in press) and Moran (Douglas & Moran, 1983; Halasz & Moran, 1982; Moran, 1983) have used such abstract cognitive constructs to predict transfer among a number of text-editing systems.

By including cognitive constructs as among the common elements that can be transferred, the cognitive approach has eliminated the old verbal learning distinction between specific and general transfer. The entire task, including aspects of the stimulus and response as well as more general or cognitive aspects, can be described or analyzed under a unified system.

The adoption of the production rule as the unit of transfer provides a quasi-standard way of describing what the common elements are. A greater contribution is their use in constructing computer models of cognitive
processes, that then can be evaluated by the standard of how well they mimic the human data. Although some people harbor an almost Luddite suspicion of such efforts, it is difficult to see how else all the elements of a complex theory can be shown to work in the manner proposed. Furthermore, such models provide a way for theorists to study the emergent behavior of their systems. For example, a test of the Soar theory/production system (Laird et al., 1984) showed that while it did very well with one type of task, with a different task it produced rampant overgeneralizations.

An important gain is the notion that problem spaces can be transferred, either borrowed whole or built out of pieces of other problem spaces. In solving a completely new problem, borrowing a familiar problem space immediately gives one access to all of the states and operators of that familiar space. This is sometimes an absolute advantage as in those cases where the new problem is an isomorph of one already known (Simon & Hayes, 1976). However, it can be a disadvantage and potential source of negative transfer when some states and operators from the borrowed problem space are completely irrelevant to the task at hand.

Another gain is in the arena of within-task transfer. The greater precision of the problem space-based learning theories such as ACT*, and Soar make it clear that many within-task transfer phenomena are actually examples of learning-as-transfer and thereby completely explained by learning theory. Thus, what was thought to be a manifestation of a higher order phenomenon, within-task transfer, is shown to be an example of a lower order phenomenon, learning.

Given all of these gains, what has been lost? We see only short-term losses. For the short-term, the field has abandoned paradigms, such as paired-associate learning, for which a wealth of empirical data exists. Eventually such paradigms will have to be subjected to a rigorous cognitive task analysis so that performance models can be created (and perhaps modeled in a production system). Such models will make two important contributions. First, they would provide an impressive demonstration of the power and generality of the cognitive science approach. Second, they would allow the existing empirical data on transfer to be interpreted in such a way as to have direct relevance to the learning and transfer of more complex cognitive skills.

A. THE ROLE OF SIMPLIFIED THEORIES FOR COGNITIVE ENGINEERING

In their many studies, Kieras, Polson, and Moran assume a static representation of cognitive skills. This contradicts what we know about how skills change with learning (see especially Pirolli & Anderson, 1985; Laird et al., 1984). It is interesting that the two research labs which make this static
representation assumption are also the two with the greatest interest in real-world applications. This is not a coincidence. Theories of learning and transfer are in an interesting intermediate state. While they can explain more data than ever before, they are not powerful enough to account for all of the interactions of the cognitive apparatus. By this view, the assumption of a static representation is a simplifying assumption that makes it possible to build useful, if limited, systems for cognitive engineering. These limited systems are more than adequate as long as their boundary conditions are well noted. Learning theories will help the cognitive engineers define what these boundary conditions are. However, in the long run, truly powerful theories and applications depend upon knowing what a person’s current knowledge representation is (this issue is clearest in the field of intelligent tutoring systems; see Gray, Mutter, Swartz, & Psopta, 1986; Sleeman & Brown, 1982). “Determining the representation a subject is using is one of the hardest problems we face and it is unclear whether the study of representation will lead the study of transfer or vice versa” (Singley & Anderson, in press).

B. THEORIES ET AL.

Writing this chapter has convinced us totally of the symbiotic relationship between theories of learning and transfer. The theoretical and practical goal of understanding the conditions of transfer of cognitive skills cannot go on in a vacuum. Before we can possibly understand what is transferred, we must understand skilled performance and how it is acquired. This remains true whether we eventually define transfer in terms of a simple common elements model or whether we decide that the entire problem space apparatus is involved in significant (and no doubt convoluted) ways.

REFERENCES


TRANSFER
OF LEARNING
Contemporary Research
and Applications

Edited by
Stephen M. Cormier
Joseph D. Hagman
Army Research Institute
Alexandria, Virginia

ACADEMIC PRESS, INC.
Harcourt Brace Jovanovich, Publishers
San Diego New York Berkeley Boston
London Sydney Tokyo Toronto