Empirical Analyses of Self-Explanation and Transfer in Learning to Program

Peter Pirolli and Kate Bielaczyc

School of Education
University of California, Berkeley

ABSTRACT

Building upon recent work on production system models of transfer and analysis-based generalization techniques, we present analyses of three studies of learning to program recursion. In Experiment 1, a production system model was used to identify problem solving that involved previously acquired skills or required novel solutions. A mathematical model based on this analysis accounts for inter-problem transfer. Programming performance was also affected by particular examples presented in instruction. Experiment 2 examined these example effects in finer detail. Using a production system analysis, examples were found to affect the initial error rates, but not the learning rates on cognitive skills. Experiment 3 examined relations between the ways in which people explain examples to themselves and subsequent learning. Results suggest that good learners engage in more metacognition, generate more domain-specific elaborations of examples, make connections between examples and abstract text, and focus on the semantics of programs rather than syntax.

INTRODUCTION

One of the classic debates in psychology has concerned the nature of the transfer of knowledge across situations of potential use (for a useful review see Singley & Anderson, 1989). One school of thought is typified by Thorndike's *theory of identical elements* (1903), which holds that transfer is a function of the stimulus-response elements acquired in one task that can be used in another task. Another school of thought is typified by Gestaltists such as Wertheimer (1945) or Katona (1940) who distinguished between *senseless* and *meaningful* learning. The Gestaltists did not deny that transfer of the kind predicted by the theory of identical elements would occur in situations of senseless learning (Singley & Anderson, 1989). However, the Gestaltists argued that transfer would be qualitatively different and superior in situations of meaningful learning, in which the learner grasped the *inner structural relationships* of the problem or task (Lewis, 1988).

The ultimate goal of the project presented here is to develop a model of the knowledge acquisition and transfer that occurs in a fairly typical lesson on programming. Here we discuss studies that suggest that transfer can be characterized by an updated version of the identical elements theory, but also that learners do differ in ways that they come to understand problems and these understandings have an impact on learning. We suggest that recent work on production system models of transfer (Singley & Anderson, 1989) and of analysis-based generalization (Lewis, 1988) may provide the basis for a model of learning and transfer that integrates the main ideas of identical elements theory and of meaningful learning.

THE LEARNING PARADIGM

Our studies focus on learning a lesson on programming recursive functions, which takes place in a longer sequence of instruction on programming. A typical programming lesson involves reading a text or listening to an instructor on some novel topic and then working through a set of relevant

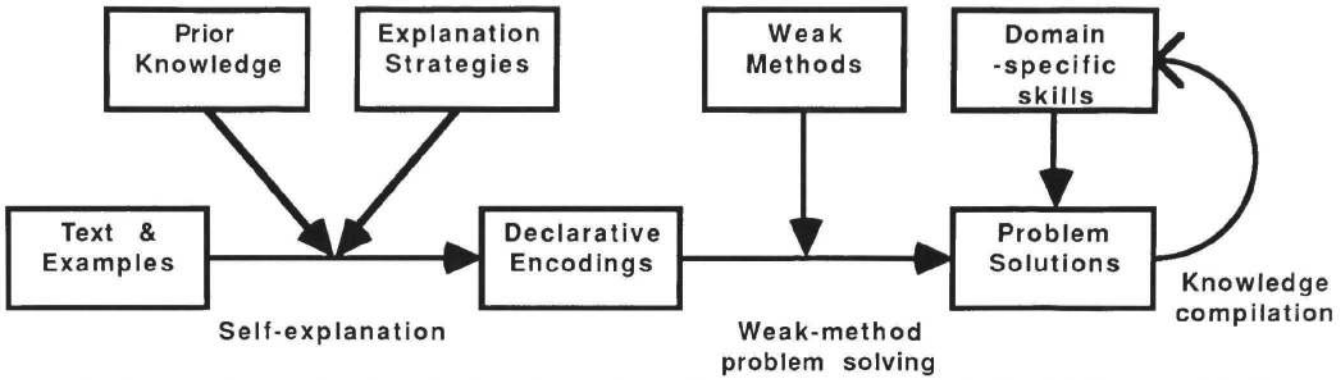

FIGURE 1: THE ANALYSIS OF INSTRUCTION AND ITS TRANSFER TO DOMAIN-SPECIFIC SKILL

exercise problems. Typically, the text or instructor will discuss some illustrative examples to facilitate learning. Figure 1 presents a simplified model of learning in a typical lesson. The boxes in Figure 1 indicate knowledge content and arrows indicate processes. In this learning situation, the learner actively constructs representations of texts and examples based on prior knowledge. This produces a set of example encodings and other relevant facts and principles that are stored as declarative knowledge in the learner's memory. Upon encountering a partially novel problem, the learner will use as much of her existing domain-specific skill as possible. At problem-solving impasses, in which no previously acquired skills are applicable, the learner resorts to weak-method problem solving. These methods operate on the declarative knowledge acquired from texts and examples. *Knowledge compilation mechanisms* (Anderson, 1987) summarize each novel problem-solving experience into new domain-specific skills.

In previous research on the acquisition of skills for programming recursive functions (Pirolli, 1986), we developed production system models of novice skill acquisition in the GRAPES production system language which emulates the skill acquisition components of the ACT* theory (Anderson, 1987). Goals are explicitly represented in GRAPES goal memory. Operators are represented by production rules that implement the basic actions available in programming (e.g., writing out a function name). Programming plans are implemented as productions that achieve goals activated in goal memory.

Such production system analyses can serve as a useful starting point in the analysis of the transfer of cognitive skill. Singley and Anderson (1989) have recently presented an ACT* theory of transfer that is in the spirit of Thorndike's identical elements theory of transfer. In its bare-bones form, the ACT* theory of transfer states that productions are the elements of transfer. Complexity is added to the ACT* analysis of transfer by considering the role of declarative knowledge. New productions are compiled as summarizations of the operation of weak methods, such as analogy, over declarative structures. Studies (Chi, Bassok, Lewis, Reiman, & Glaser, 1987; Pirolli, 1987) suggest that the effectiveness of analogy is related to the richness and content of the representations of example solutions.

Recently, Chi et al. (1987) analyzed the statements made by students learning from a physics text as they explained examples to themselves and solved a set of physics problems. Subjects were divided into groups of good and poor learners based on their problem solving performance. Good learners made significantly more elaborations of presented examples than poor students and showed greater evidence of monitoring their comprehension. In addition, there were qualitative differences in the kinds of elaborations made by good vs. poor learners with good learners showing more explanations and justifications of content relevant to subsequent problem solving.

Computational models that address the analysis of examples and subsequent generalization to novel problem solutions are called analysis-based generalization techniques by Lewis (1988). These models include production system models of analogy (Anderson & Thompson, 1986; Pirolli, 1987) and explanation-based learning methods (DeJong & Mooney, 1986; Mitchell, Kellar, & Kedar-Cabelli, 1986). One deficiency in current analysis-based generalization models is that we know little about the strategies and knowledge that learners use in constructing and using their analyses of examples. In the following studies, we present analyses of the acquisition and transfer of knowledge from instructional texts and examples to novel solutions and across problems. Our analyses build upon production system models of transfer and models of analysis-based generalization.

EXPERIMENT 1: EFFECTS OF EXAMPLES AND INTER-PROBLEM TRANSFER

Subjects ($N = 20$) in Experiment 1 proceeded through a series of programming lessons in LISP centered around an intelligent tutoring system called the LISP Tutor (Reiser, Anderson, & Farrell, 1985). For each lesson, students read some text introducing some new programming feature or technique and then worked through a set of programming problems with the LISP Tutor. The LISP Tutor instructs using a *model tracing methodology* which involves comparing a student's programming behavior to the behavior of the LISP Tutor's internal *ideal* and *buggy* models. An ideal model is a production system model of the programming skill to be acquired by subjects. A buggy model is a representation of common misconceptions and mistakes made by subjects.

Subjects were divided into groups that received a text on recursion that included either (a) an example program that worked with list inputs (list recursion example), or (b) an example program that worked with integer inputs (number recursion example). After reading their texts on recursion, subjects solved 10 recursion programming problems using the LISP Tutor. Five of these problems worked with list inputs (list problems), and the other five worked with integer inputs (number problems). Subjects were also divided into groups that received either: (a) a *blocked* sequence of problems, in which four number recursion problems were followed by four list recursion problems, with two final problems, or (b) an *intermixed* sequence, in which four number recursion problems occurred as problem trials 1, 3, 5, and 7, and four list recursion problems occurred as problem trials 2, 4, 6, and 8 (with the same final problems as the blocked sequence). For all subjects, the ordering of number recursion problems and list recursion problems was the same (although the two kinds of problems may or may not be intermixed).

Inter-Problem Transfer

Figures 2 and 3 present the mean number of errors per problem across problem trials for the intermixed and blocked sequences. An ANOVA of Sequence by Example by Problem Trial carried out on the errors per problem data revealed a main effect of Problem Trial, $F(9, 144) = 5.74$, $p < .0001$, but no main effect of Sequence, indicating that the two kinds of sequence did not produce substantially different performance overall. However, as suggested by Figures 2 and 3, there was a significant Sequence by Problem Trial interaction, $F(9, 144) = 84.38$, $p < .0001$, indicating that performance across problem trials was radically different for the two problem sequences.

According to the ACT* model of transfer of cognitive skill, the data in Figures 2 and 3 should be captured by a production system analysis that takes into account the individual productions used to solve a problem, their strength from prior practice, and the learning of new productions at the appropriate opportunities. We performed a simplified version of this analysis in which production strength was ignored, and all productions were treated as equals (i.e., we ignored variations in learning difficulties and errors rates across different productions). In this simplified model, errors

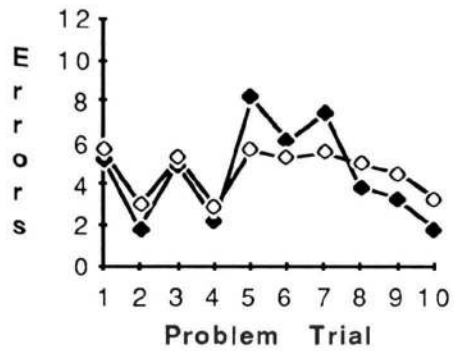

FIGURE 2:
ERRORS PER PROBLEM
FOR THE *BLOCKED* SEQUENCE

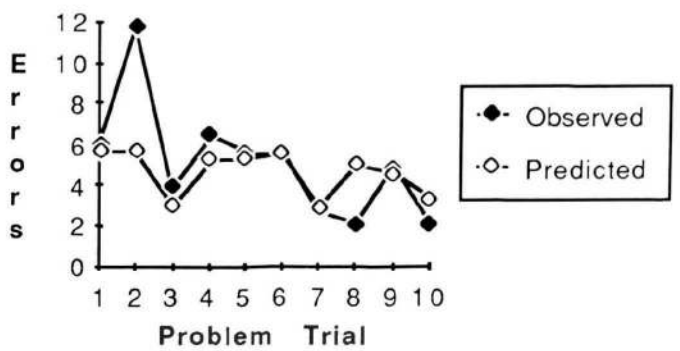

FIGURE 3:
ERRORS PER PROBLEM
FOR THE *INTERMIXED* SEQUENCE

on problem trial, t, are a linear function, $E(t)$ of the number of previously acquired productions, $P_{old}(t)$, that apply in a problem solution on trial t, and the number of novel problem solving steps, $P_{new}(t)$, for which new productions will be acquired. P_{old} and P_{new} were estimated by examining the productions used by the LISP Tutor's ideal models for the minimal program solutions across a sequence of problems. Regression of this linear model to the data in Figures 2 and 3 yields:

$$E(t) = .56\, P_{new}(t) + .14\, P_{old}(t) \qquad (1)$$

with $R = .51$. Thus a substantial proportion of the variance in Figures 2 and 3 can be captured by a simple characterization of the opportunities for the application of previously acquired productions and the places where new productions will need to be acquired.

Effects of Examples

Figure 4 presents mean errors per problem on list and number recursion problems broken down by the type of example available during instruction. Performance on problems similar to the available example is superior to performance on problems different from the example, and the interaction in Figure 4 is significant, $t(144) = 2.00$, $p < .05$.

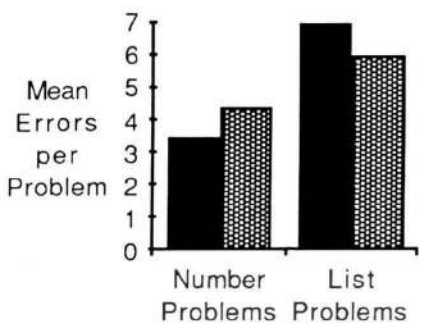

FIGURE 4:
EFFECTS OF EXAMPLES
ON PROBLEM ERRORS

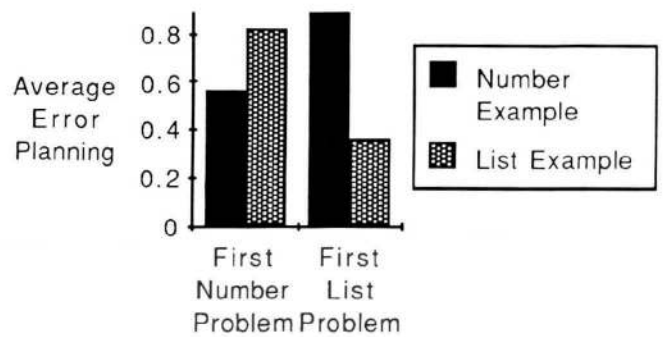

FIGURE 5:
EFFECTS OF EXAMPLES
ON PLANNING THE RECURSIVE CASES

Another measure of the impact of examples can be attained by examining a particular skill that is especially important in the task of programming recursive functions. This is the skill of *planning the recursive cases* of a recursive function. This skill involves characterizing how a function will make a recursive call to itself and use the result of that call to form an output. Figure 5 presents the LISP Tutor's diagnosis of whether or not subjects had determined a correct plan for recursive cases. The data in Figure 5 concern the first list or number problem encountered by subjects and are broken down by the kind of example presented in instruction. Again, subjects make more errors on problems that are different from the presented example. On list problems the effect is significant, Fisher p = .02, but on number problems it is only marginal, Fisher p = .18.

Summary

The results of Experiment 1 indicate that a production system analyses of the transfer of skill across problems captures a substantial amount of the performance effects as subjects progress through a sequence of problems. Interestingly, there was no effect of differences in problem sequencing as is predicted by a production system model of transfer. The examples used in Experiment 1 had clear effects on both general performance on problems and on specific skills, again in line with the general idea that knowledge is very tied to specific situations. In Experiment 2, we examine the impact of examples on the acquisition of domain-specific skills in further detail.

EXPERIMENT 2: A PRODUCTION SYSTEM ANALYSIS OF EXAMPLE EFFECTS

Subjects in Experiment 2 learned to program recursion in a simplified version of LISP without the aid of the LISP Tutor. In the target lesson on recursion, 19 subjects were presented with a text introducing recursion, and an example program was available on-line on their computer terminals. A set of 16 recursion problems and associated program solutions were created for Experiment 2. Subjects received four of these problems in a training phase, in which they received feedback for errors. The selection of examples and training problems was counterbalanced across subjects.

Example Effects

A production system model was developed in GRAPES that was capable of coding all 16 recursive functions used in Experiment 2. Of the 19 productions in this model, 16 yield some identifiable portion of code. Each attempt at coding a training program by each subject was scored for errors on the code associated with these 16 productions. Further, we identified the productions that would be used by our GRAPES model to code the example presented to each subject. On training problems, we expected subjects to show better performance on these *analogous* productions than on *nonanalogous* productions that are not used by our model in coding the example solution. This expectation is based on the assumption that subjects would have a better chance of inferring and using declarative knowledge from the example in situations involving analogous productions than in situations involving nonanalogous productions.

Figure 6 presents percent error data for analogous and nonanalogous productions over the first six opportunities for coding an action associated with a production in the training phase. The practice curves for both kinds of productions show the usual power law effects. Power functions of the form

$$P(t) = a\, t^{-b} \qquad (2)$$

are fit to the data in Figure 6, where $P(t)$ is the probability of error on trial t, a is the error rate on

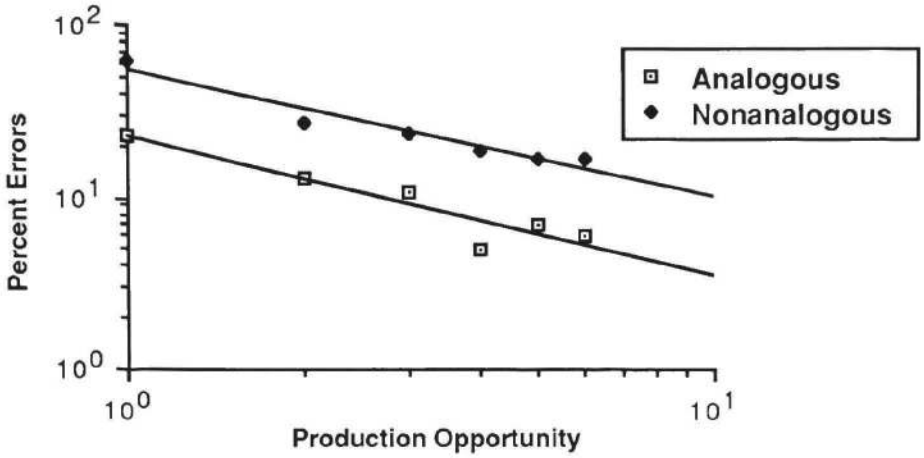

FIGURE 6: ERROR RATES AS AS A FUNCTION OF PRACTICE FOR ANALOGOUS AND NONANALOGOUS PRODUCTIONS

the first trial, and b is a rate parameter. For the analogous productions, $a = .23$ and $b = .80$, with $r = .93$. For the nonanalogous productions, $a = .55$, $b = .73$, with $r = .97$. Thus, the major difference, as indicated by differences in a, is in the error rates on the initial trials of analogous and nonanalogous productions. Given that the rate parameters, b, are relatively close for the two kinds of productions, it appears that the available example largely acts as if it were several trials of practice for the analogous productions. Thus, the results of Experiment 2 show that examples have a substantial effect on the first opportunity for acquiring a production but little or no interaction with subsequent improvement due to practice.

EXPERIMENT 3: EFFECTS OF SELF-EXPLANATION OF EXAMPLES

The model outlined in Figure 1 suggests that the manner in which subjects analyze examples will have an impact on subsequent acquisition of skill. Experiment 3 was partly modelled after the research of Chi et al. (1987). Subjects in Experiment 3 ($N = 12$) learned to program recursive functions in LISP using the LISP Tutor. While subjects were reading through their text-based instruction on recursion, we asked subjects to think out loud, and further, we asked subjects to explain all examples to themselves.

Self-explanations

Our first pass in analysis has focused on correlations between the number and kinds of self-explanations made while processing the text instruction (including examples) and subsequent performance in problem solving with the LISP Tutor. Based on the mean error rates per problem using the LISP Tutor, we performed a median split, dividing subjects into groups of good and poor learners. Verbal protocols collected while subjects read texts and examples were segmented into individual statements. Table 1 presents a summary of the mean number of *elaborations* produced by good and poor subjects as they worked through their examples. The elaborations in Table 1 are divided into different kinds. A *monitoring* elaboration refers to statements about the subjects' strategies or state of knowledge. *Activity* elaborations are comments about the instruction or the task. *Domain* elaborations concern statements about programming and recursion. The *other* category refers to incomplete phrases. Good subjects are superior to poor in all but the "other"

TABLE 1
ELABORATIONS OF EXAMPLES MADE BY GOOD AND POOR LEARNERS

	ELABORATIONS			
SUBJECTS	Monitoring	Activity	Domain	Other
Good	19.00	6.00	23.17	.17
Poor	2.50	.50	10.33	.00

category in Table 1 ($p < .05$ by t-tests). That good learners show more evidence of monitoring themselves and the instructional situations suggests higher amounts of metacognition. The greater amounts of domain-specific elaborations made by good learners also suggests that they are producing, in general, more information of potential use in later problem solving contexts.

The domain explanations given by subjects were further categorized into *syntax-oriented* statements or *semantics-oriented* statements. Syntax statements are ones that refer to the syntax of program code, or other surface features of the examples. Semantic statements are ones that provide an abstract interpretation of the process generated by a piece of code, indicate the significance of a program element, or identify the goal or purpose achieved by a piece of code. All six of the good subjects made more semantics-oriented than syntax-oriented elaborations, whereas four of the six poor subjects showed the opposite trend, an interaction significant by sign test, $p < .05$. This focus on the semantics of programming, rather than syntax, suggests that the good learners are indeed grasping the "inner structural relations" of the examples.

Connecting Examples to Text

The text of the instruction used in Experiment 3 was constructed at a fairly abstract level, with no direct references to the examples. We identified statements made while explaining the example that connected portions of the example back to concepts introduced in text. The mean number of such connections for good subjects was 4.33 and for poor subjects was .50, which is a significant difference, $t(10) = 2.18, p < .05$. The generation of connections between the examples and abstract information derived from text is the sort of process that would be predicted to be effective by analysis-based generalization methods.

GENERAL DISCUSSION

The results of Experiments 1 and 2 indicate that transfer of knowledge derived from examples to subsequent problem solving and across problem solving tasks can be substantially accounted for by production system analyses of transfer. Our current analyses in Experiment 3 indicate that complexity is added to this analysis by individual differences in the processes used in understanding instructional texts and examples. Our model in Figure 1 suggests that such differences can be attributed to differences in the prior knowledge and explanation strategies used in processing texts and examples. Current models of analysis-based generalization say little about the ways in which example analyses may vary (Lewis, 1988). In future analyses, we expect to focus on identification of process models that characterize the learning strategies of good and poor learners in programming. Although our results suggest that aspects of both the identical elements theory and the theory of meaningful learning are corroborated by our data, we do not see any reason that precludes their integration into a process model of learning and cognition.

REFERENCES

Anderson, J.R. (1987). Skill acquisition: The compilation of weak-method problem solutions. *Psychological Review, 94*, 192-210.

Anderson, J.R. & Thompson, R. (1986). *Use of analogy in a production system architecture.* Unpublished manuscript, Carnegie-Mellon University, Department of Psychology, Pittsburgh, PA.

Chi, M.T.H, Bassok, M., Lewis, M.W., Reiman, P., & Glaser, R. (1987). *Self-explanations: How students study and use examples in learning to solve problems* (Tech. Rep. 9). Pittsburgh, PA: University of Pittsburgh, Learning Research and Development Center.

DeJong, G. & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1*, 145-176.

Katona, G. (1940). *Organizing and memorizing*. New York: Columbia University Press.

Lewis, C. (1988). Why and how to learn why: Analysis-based generalization of procedures. *Cognitive Science, 12*, 211-256.

Mitchell, T.M., Kellar, R.M., & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*, 47-80.

Pirolli, P. (1986). A cognitive model and computer tutor for programming recursion. *Human-Computer Interaction, 2*, 319-355.

Pirolli, P. (1987). A model of purpose-driven analogy and skill acquisition in programming. In *Proceedings of the Cognitive Science Society Conference.*

Reiser, B.J., Anderson, J.R., & Farrell, R. (1985). Dynamic student modelling in an intelligent tutor for LISP programming. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 8-14, Los Altos, CA: Morgan-Kaufman.

Singley, M.K., & Anderson, J.R. (1989). *Transfer of cognitive skill.* Cambridge, MA: Harvard University Press.

Thorndike, E.L. (1903). *Educational psychology*. New York: Lemke & Buechner.

Wertheimer, M. (1945). *Productive thinking*. New York: Harper and Row.

ACKNOWLEDGEMENTS

This research was supported by a National Academy of Education Fellowship granted to the first author. We would like to thank Beatrice Lauman for her work in running subjects and coding data and David Rockower for implementing the GRAPES models used in Experiment 2.