DOCUMENT RESUME

ED 299 953 IR 013 466

AUTHOR Swan, Karen; Black, John B.
TITLE The Cross-Contextual Transfer of Problem Solving Strategies from Logo to Non-Computer Domains.
PUB DATE Apr 88
NOTE 17p.; Paper presented at the Annual Meeting of the American Educational Research Association (New Orleans, LA, April 5-9, 1988).
PUB TYPE Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Academic Achievement; Cognitive Mapping; Elementary Education; Hypothesis Testing; *Learning Strategies; *Problem Solving; *Programing; Psychological Studies; *Transfer of Training
IDENTIFIERS *LOGO Programing Language; *Mental Models

ABSTRACT

This report investigated the relationship between learning to program LOGO and the development of problem solving skills. Subjects were 133 students in grades 4-8 who had at least 30 hours of experience with both graphics and lists programming in Logo. Students were randomly assigned to one of three contextual groupings, which received graphics, lists, or both graphics and list problems, according to grade level. Groupings remained constant across six instructional units corresponding to six problem solving strategies believed to be helpful for children programming computers: subgoals formation, forward chaining, backward chaining, systematic trial and error, alternative problem representation, and analogical reasoning. Highly significant differences were found across both contextual groupings and grade levels for all strategies except backward chaining, suggesting that a pedagogy combining a focus on particular aspects of general problem solving, direct instruction, and a mediated learning environment will enable the development of problem solving skills with Logo programming and their transfer to non-computing domains. Highly significant differences were also found between grade levels on measures of subgoals formation, systematic trial and error, and analogy, suggesting that there are developmental differences in students' abilities to acquire and transfer particular problem solving strategies. No significant differences were found between contextual groupings, suggesting that students' abilities to transfer problem solving skills did not vary depending upon the base context(s) in which those skills were acquired. Data results are displayed in one table. (43 references) (EW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED299953

U S DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it

☐ Minor changes have been made to improve reproduction quality

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

THE CROSS-CONTEXTUAL TRANSFER OF PROBLEM SOLVING STRATEGIES FROM LOGO TO NON-COMPUTER DOMAINS

Karen Swan and John B. Black
Teachers College, Columbia University

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Karen Swan

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

RC013466

Recent nationwide assessments of student achievement have found that despite adequate basic skills, students' problem solving performance remains disturbingly poor (National Assessment of Educational Progress, 1983; Kirsch & Jungeblut, 1986; Jacobson, Doran, Chang, Humrich & Keeves, 1987). Because it uniquely combines concrete, formal and procedural representations, computer programming is commonly prescribed as fertile ground for the teaching and learning of such skills. The Logo programming language, in particular, was developed for just such a purpose (Papert, 1980). Research to date, however, has failed to clearly identify either the cognitive mechanisms or pedagogical approaches that cultivate the development of problem solving abilities *within* programming environments, let alone, their *transfer* from programming to other educational contexts (Mandinach & Linn, 1986; Clements, 1987; Pea & Kurland, 1987; Salomon & Perkins, 1987).

The research we report investigated the relationship between learning to program in Logo and the development of problem solving skills. In particular, we sought a finer-grained analysis of the problem solving strategies utilized in Logo programming environments, the pedagogical approaches that might help cultivate such processes, and the cognitive mechanisms involved in their transfer to non-computer domains. Our investigation was based on a study of the Logo/problem solving literature.

LOGO/PROBLEM SOLVING RESEARCH.

The *Syracuse Logo Project* (Statz, 1973) studied the effects of Logo programming experience on fourth graders. While reporting gains on word puzzles and permutation tasks, they found no increase in general problem solving skills. The *Brookline Logo Project* (Papert, Watt, diSessa & Weir, 1979) attempted to correlate work in Logo with increased geometric reasoning ability. Although they found some increased skills among students with Logo experience, these were not statistically significant. The *Edinburgh Logo Project* (Watt, 1982) focused on the use of Logo to create an environment for learning to think mathematically. Tests results revealed that while a Logo group improved a little more than a control group on a *Basic Maths* test, the control group evidenced slightly more improvement than the experimental group on tests of *Maths Attainment*.

Logo Project PROKOP in Darmstadt, West Germany, studied Logo-based high school mathematics programs for five years. This extensive curriculum development project experimented with the teaching and learning of problem solving skills in five areas -- non-numerical mathematics, linguistics, computer science, artificial intelligence, and gaming. Although they found significant results on computers, these did not transfer to paper and pencil tests. Similarly, Richard Noss (1984, p. 91) of the *Chiltern Logo Project* in Great Britain writes, *"Preliminary interview transcripts with children who have had 55 hours Logo experience over eighteen months, have suggested that children are capable of making use of their Logo knowledge to make interesting algebraic generalizations in a non-Logo context,"* but reports no transfer to standardized measures. The Queen's University two year study of Logo implementations in regular classrooms in Eastern Ontario (Higginson, 1982 p. 228) found that *"Logo appears to touch something quite fundamental in children's learning procedures irrespective of the 'school ability' of the child,"* but likewise no transfer to standardized measures.

Extensive studies at Bank Street College of Education in New York City (Pea & Kurland, 1984), and at the Technion in Haifa, Israel (Leron, 1985) found no

correlation at all between Logo experience and general problem solving abilities. Roy Pea writes:

> *Our psychologists studying the cognitive effects of Logo created planning tasks to reveal the development of different planning strategies, and of skills at plan revisions analogous to program revisions. In two different studies, after a year of Logo programming, these psychologists found no effects of programming on performance. Children improved with age and practice on the planning tasks, but non-programmers did just as well after a year's time as did Logo programmers.* (1984, p. 58)

A follow-up study of high school girls involved in an intensive summer mathematics program (Pea & Kurland, 1987) revealed that students were not even acquiring problem solving skills *within* the Logo programming domain.

Other studies have been more promising. Clements & Gullo (1984), for example, assessed the effects of learning Logo programming on several aspects of young children's cognition -- cognitive style, metacognitive ability, and cognitive development. Eighteen six-year-olds were randomly assigned to a Logo group or a control group receiving computer assisted instruction. The programming group scored significantly higher on post-test measures of metacognitive ability and cognitive style (reflectivity and divergent thinking), although no differences on measures of general cognitive ability were found. Miller & Emihovich's (1986) study of pre-school children's self monitoring abilities found similar significant increases in children's ability to detect embedded errors during a referential communication task. In an interesting follow-up (Clements, 1987), the same children were again tested eighteen months following their Logo or CAI experience. On the *Test of Cognitive Skills*, a strong treatment effect was found on the analogies subtest, and on the *California Achievement Test* strong treatment effects were found on subtests of language mechanics and reading vocabulary.

At the Lamplighter School in Dallas, Texas, students, were given a rule learning task in which they were shown a series of cards displaying simple figures varying in shape, size, number, and color. Students were told what attributes to attend to, then asked to determine the rule governing the selection of the cards. Researchers found that third graders with extensive Logo experience scored significantly higher on the test than did another third grade group with less Logo experience, and higher than *sixth* graders with no Logo experience (Gorman & Bourne, 1983). Recent research on the ability of kindergarten children to solve affirmative and conjunctive rule-learning tasks (Degelman, Free, Scarlato, Blackburn & Golden, 1986) supports these findings.

Littlefield, Delclos, Franks, Clayton, and Bransford (1986) studied the effects of various teaching methods on students' acquisition of both Logo programming and general problem solving skills. They found that while students working in a discovery learning environment acquired neither, students receiving direct instruction at least learned Logo programming concepts. Students receiving direct instruction in mediated learning environments, moreover, both acquired Logo programming skills and showed an increased ability to solve simple mapping problems. None of the instructional groups, however, improved on general problem solving measures.

Sharon Carver (1987) used an extensive task analysis of Logo debugging skills, and pilot research with second graders learning Logo in a traditional environment, to devise an intervention focused on the development of such skills using mediated learning techniques. She then tested the intervention on fourth graders. She reports that students both acquired Logo debugging skills and transferred them to a non-Logo task involving debugging faulty directions.

A review of the Logo/problem solving literature, then, presents a mixed picture of the usefulness of the language for the teaching and learning of problem solving. Three pedagogical elements, however, seem common to successful interventions. These are *a focus on particular aspects of general problem solving ability* (Statz, 1973; Gorman & Bourne, 1984; Clements & Gullo, 1984; Clements, 1986; Miller & Emihovich, 1986; Degelmen, *et al.*, 1986; Littlefield, *et al.*, 1987: 1986; Carver, 1987), *direct instruction* (Boecker & Fisher, 1982; Littlefield, *et al.*, 1986; Carver, 1987), and *the use of mediated learning techniques in teacher/student interactions* (Clements & Gullo, 1984; Clements, 1986; Miller & Emihovich, 1986; Littlefield, *et al.*, 1986; Carver, 1987). Seen in this light, the Logo/problem solving literature suggests that a pedagogical approach incorporating those elements will support the transfer of problem solving skills from Logo programming to non-computing contexts. The research we report investigates that pedagogy.

PROBLEM SOLVING STRATEGIES STUDY.

Problem solving can be decomposed into a number of distinct strategies (Polya, 1973; Wicklegren, 1974). Certain of these seem more applicable to programming problems in general, children's programming in particular (Clements & Gullo, 1984; Lawler, 1985; Clement, Kurland, Mawby & Pea, 1986). Indeed, much of the more successful Logo research has focused on particular aspects of the problem solving processes -- cognitive style (Clements & Gullo, 1984), metacognitive ability (Miller & Emihovich, 1986; Clements, 1987), rule learning (Gorman & Bourne, 1983; Degelman, *et al.*, 1986), mapping (Littlefield, *et al.*, 1986), debugging (Carver, 1987). Correspondingly we focused our research on particular problem solving strategies. We identified six particular problem solving strategies we believed might be useful to children programming computers at some point in their development. These were *subgoals formation*, *forward chaining*, *backward chaining*, *systematic trial and error*, *alternative problem representation*, and *analogical reasoning*.

Subgoals formation.

Subgoals formation refers to breaking a single difficult problem into two or more simpler problems. Even when no obviously solvable subgoals can be found, breaking a problem into parts makes its solution seem less formidable, more manageable, and less susceptible to errors. We analyzed the process of subgoals formation into the following four steps:

1. *Problem definition.* Specify the problem.
2. *Subdivision.* Examine the problem specification to see where it can be broken into smaller, self-contained problems. Specify these and their connections to the larger problem.
3. *Evaluation.* Test the subproblems generated for grain size and further decomposition. If the subproblems are manageable or cannot be further decomposed, solve them. Recombine these

partial solutions into the total solution using the connections specified in step 2.
4. *Recursion.* Otherwise, repeat the second and third steps for each of the subproblems generated. Continue the process until no more smaller problems can be generated for any of the subproblems.

While subgoals formation might seem an obvious strategy to adults, it is not at all obvious to many children (Carver and Klahr, 1986). Of all the problem solving strategies we have isolated, however, it can most clearly be implemented and concretized in the Logo environment through structured programming techniques. In Logo, small subprocedures are easily written and placed in the workspace. Because these can be called from anywhere in a program, a program can simply be a list of subprocedures, a very concrete representation of the subgoals that make up a programming solution.

Forward chaining.

Forward chaining involves working from what is given in a problem towards the problem goal in step-by-step, transformational increments that bring one progressively closer to that goal. The forward chaining process can be decomposed into the following steps:

1. *Problem definition.* Specify the problem goal. Specify what is given. Specify the constraints, if any.
2. *Transformation.* Use domain operators to manipulate the givens to bring them closer to the goal state.
3. *Evaluation.* Compare the desired goal, the givens, and the transformation. Test to see whether the transformation is really closer to the goal than the givens. If it is not, redo step 2.
4. *Recursion.* Make the transformation a new given. Repeat steps 2, 3 and 4 using the new given. Continue in this manner until the goal state is reached and the problem is solved.

A programming environment, especially an interpreted environment like Logo, is inherently supportive of the forward chaining process. Transformations can be implemented, their effects accessed, and successful changes instantiated as partial programs, with relative ease and little risk. A program can thus be developed in incremental steps and such development provide a concrete model of the forward chaining process. An important part of forward chaining, however, involves the ability to choose appropriate transformations and evaluate whether or not these actually bring one nearer problem solution. Forward chaining thus requires at least some sort of mental model of the problem space, and is not, therefore, typically a novice technique. We believe, therefore, that subjects' domain expertise may effect their ability to apply forward chaining strategies within the Logo environment, and correspondingly, significantly effect its transfer to non-computer contexts.

Backward chaining.

Backward chaining, as opposed to forward chaining, is typically a novice problem solving strategy. Backward-chaining focuses on the goal state and tries to deduce a preceding state from which that goal could be derived, then a state from which *that* state could be derived, and so on, working backward to what is given in a problem. Our analysis of the backward chaining process consists of the following four steps:

1. *Problem definition.* Specify the goal state. Specify what is given in the problem.
2. *Decomposition.* Specify a state that could be transformed into the goal state in a single step.
3. *Evaluation.* Test the specified transformation to be sure it can in fact be transformed into the goal state. If it cannot, redo step 2. Examine the specified transformation to make sure it is closer to what is given than is the goal. If it isn't, redo step 2.
4. *Recursion.* Otherwise, make the specified state a new goal state and repeat steps 2 and 3 using this new goal state. Continue in this manner until a clear path from givens to goal can be discerned. Use it to solve the problem.

Backward chaining is a particularly useful technique when a problem has a uniquely specified goal, and/or is a situation in which the inputs and outputs of the transformations involved in going from goal to givens can be uniquely specified (Wicklegren, 1974). Such is often the case in programming problems. Indeed, programming has been identified as a teleological domain (Bolter, 1984). Moreover, in as much as it is typically a novice technique, backward chaining might be more available to children.

Systematic trial and error.

Systematic trial and error involves the recursive testing of possible solutions in a systematic, guided fashion, and the problem reduction and/or refinement resulting from such tests. Our analysis of the systematic trial and error process includes:

1. *Problem definition.* Specify the problem goal.
2. *Approximate solution.* Create and implement a plan to solve the problem.
3. *Evaluation.* Compare the problem goal with the instantiated solution. If there are no discrepancies between them, the problem is solved. Otherwise, generate a description of the discrepancies between the desired goal and the instantiated solution.
4. *Recursion.* Use the description of goal/solution discrepancies to revise the plan, and reapply step 2 and 3. Continue in this manner until the instantiated solution matches the desired goal.

Piaget (1971) believed that the application of systematic trial and error strategies was an important determinant of formal operational ability. Systematic trial and error, then, is an obvious candidate for testing Papert's (1980) notion that programming environments support the concretizing of the formal. Certain types of graphics programming, moreover, are paradigmatic of systematic trial and error strategies. The creation of a drawing, for example, requires the progressive testing and refinement of its procedures. Debugging also makes use of, and provides symbolic representations for, such techniques (Carver, 1987).

Alternative Representation.

Alternative representation involves conceptualizing a problem from differing perspectives. Polya (1973) writes that often the way a problem is stated is really all that makes it difficult. Simple restatement will make its solution obvious. We

have developed the following four-step description of the development of alternative representations of a problem:

1. *Problem definition.* Specify the problem.
2. *Alternative representation.* Generate an alternative problem specification.
3. *Evaluation.* Test to see whether the new problem specification suggests problem solution. If it does, solve the problem.
4. *Recursion.* Otherwise, repeat the second and third steps by generating and evaluating another problem specification.

We believe programming is conducive to the development of alternative representations both because there are never single correct solutions to programming problems, and because differing representations can quite easily be instantiated and pragmatically tested within programming environments. The cooperative atmosphere of the typical Logo classroom, lends additional support to the development of differing problem solutions. Logo teachers will often point to differing ways of solving a problem in the language. Indeed, Clements and Gullo's (1984) study of the effects of Logo programming on young children's cognition found significant increases in the ability to produce alternative representations among children with Logo experience. Statz's (1973) finding of significant increases on permutation tasks may also support this view.

Analogy.

Analogy involves the discovery of a particular similarity between two things otherwise more or less unlike, "*a mapping of knowledge from one domain (the base) into another (the target) predicated on a system of relations that holds among the objects of both domains.*" (Gentner, 1987). An important factor in this process, especially in problem solving contexts, is goal-relatedness, how one thing is like another with respect to a specified goal (Holyoak, 1985). The use of analogy in problem solving can be decomposed into the following steps:

1. *Problem definition.* Specify the desired goal. Specify the base and the target systems.
2. *Mapping.* Perform a mapping between the base and target systems.
3. *Evaluation.* Test the soundness of the match in terms of both structural similarity and pragmatics (goal related conditions). If the analogy generated meets the goal conditions, and the structural similarity between the base and the target holds, the mapping is sound. Solve the problem.
4. *Recursion.* Otherwise, return to step 2 and generate another and evaluate it (step 3). Continue in this manner until an adequate representation is discovered.

We believe that programming environments inherently support the development of analogy, in that one is always mapping between computer code (a formal representation) to program output (a concrete representation). Indeed, Doug Clements' (1987) study found significantly better analogical reasoning among students with prior Logo experience.

RESEARCH HYPOTHESES.

These six problem solving strategies, then, represented our best guess as to what aspects of general problem solving ability might be most applicable to children

programming in Logo. In an effort to isolate the particular techniques most relevant in Logo programming and/or most available for transfer to other domains, we developed our instructional units and our measuring instruments around these six strategies.

A second notion guiding our research was that the teaching and learning of Logo has been notoriously non-directive (Leron, 1985; Pear & Kurland, 1987; Salomon and Perkins, 1987). It is hardly likely, we felt, that the transfer of any complex skill is so automatic. Indeed, positive results have been reported from Logo interventions involving structured, as opposed to non-directed, learning designs (Miller & Emihovich, 1986; Littlefield, *et al.*, 1986; Carver, 1987; Clements, 1987; Salomon & Perkins, 1987). We determined, therefore, to focus attention specifically on the problem solving strategies we identified. We devised introductory, off-computer activities to highlight these particular techniques, and followed these with sets of programming problems particularly amenable to solutions employing such strategies.

A third consideration involved the importance of metacognitive monitoring in the problem solving process. We thought that an explicit focus on the cognitive processes involved in each of the particular problem solving strategies we identified would help children to understand and internalize such metacognitive behaviors. Although the paired work on computers and peer-teaching that are typically an important part of Logo classrooms support such focus, we felt the modeling by teachers of the cognitive processes involved in solving programming problems was indicated.

Indeed, what Feuerstein (1980) calls *mediated learning*, has been demonstrated to be an effective means for teaching self regulated learning in general (Kendall & Hollon, 1979; Meichenbaum, 1977) as well as subject matter specific learning strategies (Bereiter & Bird, 1985; Schunk, 1984; Palinscar & Brown; 1984; Weinstein & Mayer, 1986; Corno, 1986). It stands to reason that the use of similar approaches in the teaching and learning of problem solving in Logo could support the development of the important metacognitive components of problem solving behaviors within Logo environments as well. We therefore determined to use a mediated learning approach in our instructional interventions.

Our first hypothesis, then, was that given focused instruction and practice in a mediated learning environment in applying each of the identified problem solving strategies to the solution of Logo programming problems, students would transfer and apply these to problem solutions in non-computing domains. That is, we felt that the combination of three instructional components -- a focus on particular aspects of general problem solving, direct instruction, and a mediated learning environment -- would support the development of problem solving skills within Logo programming contexts and their transfer to non-computing domains.

Our second hypothesis was that there are developmental differences in students' ability to acquire and transfer problem solving skills. It seemed to us likely that problem solving skills are themselves instances of a larger ability to think logically and abstractly, and that this ability, which Piaget (1971) terms *formal operational*, develops slowly during the period when a child is in middle and junior high school. We thought, therefore, that students might be more likely to acquire and transfer problem solving strategies at differing grade levels. Our research was thus directed at children in this grade range to try and determine whether and when Logo experience might be most useful.

Finally, the teaching and learning of Logo has been, for the most part, restricted to the *turtle graphics* domain generally associated with the language. The Logo language, however, is far richer than turtle graphics alone would suggest. Logo make *list manipulation* accessible to young children (Swan, 1986), and programming with lists invokes a context quite different from graphics, a context which, we believe, more faithfully evokes the workings of the language itself. There is reason to believe that such deep structural understanding is an important factor in the development of problem solutions (Greeno & Simon, 1984). Additionally, research by Gick and Holyoak (1983) suggests that the transfer of problem solutions more readily occurs when these are initially learned in more than one context.

Gick and Holyoak were interested in the transfer of problem solutions embedded in story contexts. They presented subjects with Duncker's (1945) classic *radiation problem* -- how to use radiation beams to destroy a tumor in a patient without destroying the patient. The solution is to use several less harmful beams that converge and sum to destructive potency only on the tumor. Some of the subjects had previously read a story about a general who broke up his forces to avoid alerting an enemy to his attack, converging them for the successful conquest of an enemy fortress. Gick and Holyoak hypothesized that subjects who had read the military story would transfer the embedded strategy and so be more likely to solve the radiation problem. They were wrong. All subjects experienced equal difficulty solving the radiation problem. They found, however, that given a second story illustrating the divide and conquer strategy, subjects were better able to solve the radiation problem.

It seems that learning is context dependent. Our minds have no way of distinguishing what *is* from what *is not* significant in any given situation, thus, we tend to store information as an undifferentiated whole. When an idea is encountered in two or more contexts, however, context drops away and ideas are more clearly delineated. General concepts are abstracted from similarities among individual cases; problem solving techniques are generalized from particular problem solutions. We thought that such might be the case with the transfer of problem solving from Logo contexts. We thought it possible that students applying problem solving strategies to the solution of both graphics and lists problems might be more likely to transfer them to the solution of problems in non-computing domains.

We created three student groupings -- students working with just graphics programming problems, students working with just list processing problems, and students working with both graphics and list programming problems. We varied the contexts of the problems which they were given, to distinguish between the efficacies of these varying base domains. Our third hypothesis, then, was that there would be differences in students' abilities to transfer problem solving skills depending on the base context(s) in which those skills were acquired. In particular, we thought that if the concreteness of the base domain was the dominant factor, students programming in just graphics would show the greatest transfer effects; if a clear model of the problem space was the dominant factor, students working with just lists problems would show the greatest transfer effects; and that if multiple base domains was the most important factor, students working with both graphics and lists problems would show the greatest transfer effects.

METHOD.

Subjects.

Our subjects were 133 students in the fourth through eighth grades of a private suburban elementary school taken from their regular computer classes. All students had at least 30 hours previous experience with both graphics and lists programming in Logo.

Procedure.

All subjects were pre-tested on their ability to solve problems requiring the use of each of the six problem solving strategies. They were then randomly assigned by grade to one of the three contextual groupings, receiving respectively graphics, lists, or both graphics and lists problems. Groupings remained constant across the six instructional units corresponding to the six identified problem solving strategies. A consistent instructional sequence was followed for each strategy unit. Upon completion of all six units, subjects were post-tested using different but analogous non-computing problems. Differences between pre- and post-test mean scores were examined using analysis of variance. The study thus involved a six (strategies) by five (grade levels) by three (contextual groupings) design. Independent variables were grade level, strategy, and contextual grouping. The dependent variables were the scores on the tests of each of the problem solving strategies.

Intervention.

Students were introduced to each problem solving strategy through whole group activities designed to provide concrete, off-computer models of the cognitive processes involved in them. Forward chaining, for example, was introduced with a *treasure hunt* in which students followed a sequence of clues to discover a hidden treasure. Classic puzzles whose solution involved the application of the particular strategy under discussion were also solved and discussed with the whole class, culminating in a discussion of the cognitive processes involved in the application of the strategy.

The group work of these introductory exercises was followed by individual or paired work on unit problem sets comprised of four programming problems each. These were varied according to student groupings -- students in the *graphics* condition were given just graphics problems to work on, students in the *lists* condition were given solely list processing problems, and students in the *two-domain* condition were given both graphics and lists problems to solve.

Students worked on problems during two 45-minute class periods per week for approximately 12 weeks. A teacher and/or an intern were available for help on all problems. Both maintained a *mediated learning* approach toward student assistance, eliciting student and/or modeling their own cognitive processes as they guided students toward problem solution. For each programming problem they solved, students were required to fill out a problem solving worksheet that showed the *givens*, the *goal*, and the *solution steps* for each problem, and to turn in a listing and a run of their program.

Tests.

Subjects were tested both before and after the entire, six-unit intervention on their facility in applying each of the six identified problem solving strategies. We designed a separate measure for each of these:

Subgoals formation. Our measure of students' ability to decompose complex problems into smaller subgoals units consisted of mathematical word problems that required decomposition for correct solution. Students were asked not only to solve the problems but to show how they broke them into parts. They were given credit for correctly identified subgoals, as well as for the correct answer.

Forward chaining. The test designed to measure subjects' forward chaining skills was a paper-and-pencil version of the computer program *Rocky's Boots* (The Learning Company, 1982). In *Rocky's Boots*, symbolic *and*, *or*, and *not* gates are combined to produce machines that respond to targeted attributes and sets of attributes (*e.g.* blue diamonds, crosses or green circles, etc.). Combinations of gates must be built up in a forward chaining manner to achieve correct solutions. Our paper and pencil version had subjects draw the required connections.

Backward chaining. The test designed to measure subjects' backward chaining skills was a paper-and-pencil adaptation of the computer game *The Factory* (Sunburst, 1984). In *The Factory*, players are shown a finished product and asked to combine various machines to produce a similar product. Thus, players must work backwards from the product to deduce a correct sequence of machines that will produce it. Our paper-and-pencil version had subjects list the required machine sequence.

Systematic trial and error. Cryptography involves systematically trying and testing different symbol combinations to attain coherent decoding systems. We chose two decoding exercises to test subjects' abilities to systematically utilize trial and error strategies. The first of these was a shifted alphabet code. The second involved variations on a number code problem from Newell and Simon (1971):

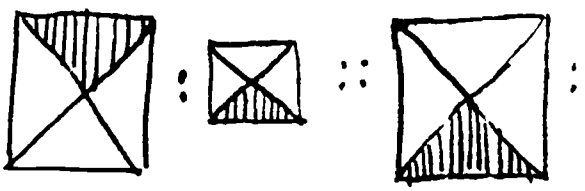
$$\begin{array}{r} \text{DONALD} \\ +\ \text{GERALD} \\ \hline \text{ROBERT} \end{array} \qquad ;D = 5$$

Alternative representation. The measure of students' ability to create alternative representations we used, was derived from the figures subtest of the *Torrance Test of Creative Thinking* (Torrance, 1972). Students were given sets of either parallel lines or circles and asked to use these as a basis for producing as many interesting and unusual drawings as they could make.

Analogy. Subjects' skill at analogical reasoning was measured with completion exercises comprised of items representing both verbal and visual analogies. Verbal items were of the following form:

SUGAR : SWEET :: LEMON :

Visual analogies were similar:



Different but analogous problems were given on the pre-and post-tests for each technique, and differences between the two examined using analysis of variance.

RESULTS.

Highly significant ($p < .001$) differences between pre- and post-test scores were obtained on measures of all strategies except backward chaining, across both contextual groupings and grade levels. These results support our first hypothesis, that a pedagogy combining a focus on particular aspects of general problem solving, direct instruction, and a mediated learning environment will enable the development of problem solving skills within Logo programming contexts and their transfer to non-computing domains.

MEAN SCORES

SUBGOALS FORMATION

	4th	5th	6th	7th	8th	total
pre-	4.3	6.9	8.2	10.9	10.6	7.5
post-	6.3	8.4	10.6	12.6	12.6	9.8

FORWARD CHAINING

	4th	5th	6th	7th	8th	total
pre-	8.4	7.7	7.8	10.9	8.6	8.4
post-	9.7	9.8	10.5	11.5	11.4	10.4

BACKWARD CHAINING

	4th	5th	6th	7th	8th	total
pre-	11.0	10.3	11.4	10.9	11.2	10.9
post-	10.8	10.9	10.6	11.5	12.4	11.2

SYSTEMATIC TRIAL AND ERROR

	4th	5th	6th	7th	8th	total
pre-	0.4	5.7	6.0	6.8	10.4	5.1
post-	7.6	8.0	10.6	11.4	13.1	9.5

ALTERNATIVE REPRESENTATION

	4th	5th	6th	7th	8th	total
pre-	92	96	91	108	94	96
post-	115	133	143	175	134	136

ANALOGY

	4th	5th	6th	7th	8th	total
pre-	17.5	18	18.8	19.4	19.2	18.4
post-	17.9	20.1	19.6	21.3	21.1	19.5

We also found highly significant ($p < .01$) differences between grade levels on measures of subgoals formation, systematic trial and error, alternative representation, and analogy, supporting our second hypothesis, that there are developmental differences in students' abilities to acquire and transfer particular problem solving strategies. Younger students exhibited greater increases on tests of subgoals formation and systematic trial and error; older students had greater gains on alternative representation and analogy measures. These results seem to indicate differential readiness to acquire specific problem solving strategies.

No significant differences between contextual groupings was found ($p > .1$). Our third hypothesis, that students' abilities to transfer problem solving skills would vary depending on the base contexts(s) in which those skills were acquired, was thus not confirmed. It may be that more than two, or that two more different, domains are necessary before the effects of multiple domains can be realized.

In future research, we plan to investigate how and how much the various aspects of our current intervention design contributed to the transfer effects we found. We also intend to investigate the specific cognitive and metacognitive mechanisms involved in the learning and transfer of problem solving strategies from Logo programming domains. In particular, we want to determine whether computer use makes a critical contribution, that is, whether, as Papert (1980) suggests computer manipulatives have something to offer to the transfer process over and above concrete manipulatives.

References

Bereiter, C. and Bird, M. (1985) Use of thinking aloud in identification and teaching of reading comprehension strategies. *Cognition and Instruction, 2,* 131-156.

Boecker, H. D. and Fischer, G. (1982) Logo Project PROKOP. *Byte, 7,* (8), 329-330.

Bolter, J. D. (1984) *Turing's Man.* Chapel Hill, NC: University of North Carolina Press.

Carver, S. M. (1987) Transfer of Logo debugging skill: analysis, instruction, and assessment. *Computer Systems Group Bulletin, 14,*(1), 4-6.

Clement, C. A., Kurland, D. M., Mawby, R. & Pea R. D. (1986) Analogical reasoning and computer programming. *Journal of Educational Computing Research, 2,*(4), 73-94.

Clements, D. H. (1987) Longitudinal study of the effects of Logo programming on cognitive abilities and achievement. *Journal of Educational Computing Research, 3,*(1), 73-94.

Clements, D. H. & Gullo, D. F. (1984) Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76,* 1051-1058.

Corno, L. (1986) Teaching and self-regulated learning. Paper presented at the annual meeting of the American Educational Research Association. San Francisco.

Degelman, D., Free, J. U., Scarlato, M., Blackburn, J. M. and Golden, T. (1986) *Journal of Educational Computing Research, 2,*(2), 199-205.

Duncker, K. (1945) On problem solving. *Pyschological Monographs, 58.*

Ehrlich, K., Soloway, E. and Abbott, V. (1982) *Transfer Effects from Programming to Algebra Word Problems: A Preliminary Study.* (Research Report #257). New Haven, CT: Yale University Department of Computer Science.

Gick, M. L. and Holyoak, K. J. (1983) Schema induction and analogical transfer. *Cognitive Psychology, 12,* 306-355.

Gorman, H., Jr. and Bourne, L. E. (1983) Learning to think by learning Logo: rule learning in third-grade computer programmers. *Bulletin of the Psychonomic Society, 21,* 165-167.

Greeno, J. G. & Simon, H. A. (1984) *Problem solving and reasoning,* (Technical Report No. UPITT/LRDC/ONR/APS-14). Washington, DC: Learning Research and Development Center, Office of Naval Research.

Higginson, W. (1984) About that rose garden: remarks on Logo, learning, children and schools. *Pre-Proceedings of the 1984 International Logo Conference,* 31-38, Cambridge, MA.

Howell, R. D., Scott, P. B. and Diamond, J. (1987) The effects of the "Instant" Logo computing language on the cognitive development of very young children. *Journal of Educational Computing Research, 3,* (2), 249-260.

Jacobson, W. J., Doran, R. L., Chang, E. Y. T., Humrich, E. Keeves, J. P. (1987) *The Second IEA Science Study -- United States.* New York: Teachers College, Columbia University.

Kendall, P. C. and Hollon, S. (Eds.) (1979) *Cognitive-Behavioral Interventions: Theory, Research, and Procedures.* New York: Academic Press.

Kirsch, I. S. and Jungeblut, A. (1986) *Literacy: Profiles of America's Young Adults.* (Report No. 16-pl-02). Princeton, NJ: Educational Testing Service.

Lawler, R. W. (1985) *Computer Experience and Cognitive Development: A Child's Learning in a Computer Culture.* New York: Halsted Press.

Leron, U. (1985) Logo today: vision and reality. *The Computing Teacher, 12,*(6), 26-32.

Mandinach, E. B. and Linn, M. C. (1986) The cognitive effects of computer learning environments. *Journal of Educational Computing Research, 2,*(4), 411-428.

Meichenbaum, D. (1977) *Cognitive Behavior Modification.* New York: Plenum.

Miller, G. E. and Emihovich, C. (1986) The effects of mediated programming instruction on preschool children's self-monitoring. *Journal of Educational Computing Research, 2,* (3), 283-297.

National Assessment of Educational Progress. (1983) *The Third National Mathematics Assessment: Results, Trends and Issues,* (Report No. 13-MA-01). Denver, CO: Educational Commission of the States.

Newell, A. and Simon, H. A. (1972) *Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall.

Noss, R. (1984) Explorations in mathematical thinking: some implications from Logo classrooms. *Pre-Proceedings of the 1984 International Logo Conference,* 85-91, Cambridge, MA.

Palinscar, A. S. and Brown, A. L. (1984) Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction, 1,* 117-175.

Papert, S. (1980) *Mindstorms.* New York: Basic Books.

Papert, S., Watt, D. diSessa, A, & Weir, S. (1979) *Final Report of the Brookline Logo Project,* (Logo Memo 53). Cambridge, MA: MIT Artificial Intelligence Laboratory.

Pea, R. D. (1984) Symbol systems and thinking skills: Logo in context. *Pre-Proceedings of the 1984 International Logo Conference,* 85-91, Cambridge, MA.

Pea, R. D. and Kurland , M. K. (1984) On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*,(2), 137-167.

Pea, R. D. and Kurland, D. M. (1987) Logo programming and the development of planning skills. In K. Sheingold and R. D. Pea (Eds) *Mirrors of Minds*. Norwood, NJ: Ablex Publishing.

Piaget, J. (1971) *Genetic Epistemology*. New York: W. W. Norton.

Polya, G. (1973) *How To Solve It*. Princeton, NJ: Princeton University Press.

Salomon, G. and Perkins, D. N. (1987) Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research, 3*,(2), 149-170.

Schunk, D. H. (1984) Self-efficacy perspective on achievement behavior. *Educational Psychologist, 19*, 848-857.

Statz, J. (1973) *Problem Solving and Logo: Final Report of the Syracuse Logo Project*. Syracuse, NY: Syracuse University.

Swan, K. (1986) Primarily lists. *PreProceedings of the 1986 International Logo Conference*. Cambridge, MA.

Torrance, E. P. (1972) *Torrance Tests of Creative Thinking*. Lexington, MA: Personal Press.

Watt, D. (1982) Logo in the schools. *Byte, 7*, (8), 116-134.

Weinstein, C. E. and Mayer, R. E. (1985) The teaching of learning strategies. In Wittrock, M. C. (Ed.) *Third Handbook of Research on Teaching*. New York: Macmillan Co.

Wickelgren, W. A. (1974) *How to Solve Problems*. San Francisco: W. H. Freeman.