POSIT: Process Oriented Subtraction-Interface for Tutoring

by

Michael Andrew Orey III

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree

of

Doctor of Education

in

Curriculum and Instruction

Approved:

John K. Burton, Chairman

Catherine A. Brown

Terry M. Wildman

James W. Garrison

John W. Roach

June, 1989

Blacksburg, Virginia

POSIT: Process Oriented Interface for Subtraction Tutoring

by

Michael Andrew Orey III

Committee Chairman: John K. Burton
Curriculum and Instruction

(Abstract)

The purpose of this dissertation was to design, develop and field test an Intelligent Tutoring System (ITS) which I have called Process Oriented Subtraction-Interface for Tutoring or POSIT. POSIT is an Intelligent Tutoring System, developed on a microcomputer, and based on Anderson's (1982, 1987) ACT* model of learning. Unlike the tutoring systems that were developed by Anderson and his colleagues (Anderson, Boyle & Reiser, 1985; Anderson & Reiser, 1985) which focussed on the tutoring of students in the context of problem solving, this system focuses on the tutoring of a cognitive skill - subtraction of whole-numbers. Because ACT* theory explicitly describes the interaction of declarative and procedural knowledge (procedural knowledge is dependent on declarative knowledge), this learning theory is ideally suited to the learning of a specific procedure.

Further, other "intelligent" systems which have been applied to subtraction (Brown & Burton, 1978; Ohlsson & Langley, 1985; Young & O'Shea, 1981) tend to focus on the answers to subtraction exercises (product oriented). POSIT, on the other hand, is an interactive system that determines errors made by the child as the child attempts to solve subtraction problems. Another difference with previous systems is that POSIT has a teaching component. Other systems assume that instruction has been given at some other point in time prior to the use of the system.

One final difference between POSIT and other systems is that it was developed with consideration of the diverse student population that is found in schools. Design decisions were based on the cognitive needs of low-, average- and high-achieving students. Such notions as reading level and complexity of the task were considered with regard to low achieving students. POSIT's ability to allow for a wide variety of algorithms was a consideration with regard to high achieving students and would also benefit students from all levels.

The results of the field test of POSIT indicate that the error model used by POSIT was very successful (76% accurate, with potential to improve to between 80 to 90%). In addition, students appear to learn from the system as measured both on the system as well as on a paper and pencil transfer test.

Acknowledgements

I would like to begin these acknowledgements with my family. First, my Mom and Dad's support throughout my education process, no matter which path I chose, was unwavering. My sister , who offered me her home on vacations; and, my sister , who always knew the right questions to run-up my Dad's phone bill, I thank. My brother and my sister-in-law , who were always more productive than me, I also wish to thank. As the result of their productivity, I would like to acknowledge my love and affection for their children , and a to-be-named child in the waiting womb. Finally, skipping back a few generations, I would like to thank my Grandma for her vigilance and concern for my welfare.

I would also like to thank my dissertation committee for their insights and criticisms during the development of this dissertation -- Cathy Brown for her perspectives from mathematics education; John Roach for his perspectives from artificial intelligence; Terry Wildman for his perspectives from educational psychology; and, James Garrison who turned my world upside down and gave new meaning to the concept of "milk and cookies."

My advisor, John Burton, whose time and patience were pushed to the limit by my educational development I would also like to acknowledge. Further, his interest in making the advisor/advisee relationship more collegial resulted in a lasting friendship. I leave Virginia Tech wondering how I can become half the advisor he is.

I would also like to acknowledge the assistance of . I elected the archaic method of paper and pencil to write this document. took the time to convert the mess I created in the dark of the night into a readable form. Without her assistance and friendship, I could not have completed this task. In addition, I would like to thank the folks in the microcomputer lab who played a role in the development of my thinking. Those folks are: , , , , , , , , and .

Finally, I would like to thank a group of friends in California who did their best to shake the foundations of my Ivory Tower -- , ,

and the most recent arrival, .

Table of Contents

List of Tables

List of Figures

POSIT: Process Oriented Subtraction-Interface for Tutoring

Introduction

The purpose of this dissertation is to design, develop and evaluate an Intelligent Tutoring System (ITS) on a microcomputer which can be placed in a regular classroom to meet the needs of the students who have a large range of abilities (whether they have learning problems or not). I have called this system Process Oriented Subtraction-Interface for Tutoring or POSIT. The focus is at the elementary level. The content is multi-digit whole-number subtraction, an area already explored by AI researchers (see, eg., Brown & Burton, 1978; Ohlsson & Langley, 1985; Young & O'Shea, 1981). The evaluation, described in the section entitled Method of Evaluation, was of two kinds: formative, whether the text used for instruction was understandable, whether a child could use the interface and whether the system was error free; and, summative, whether the system correctly determines errors and whether the instruction was effective.

POSIT is an interactive, ITS for subtraction of whole numbers. One of its advantages over previous diagnostic systems for subtraction (eg., Brown & Burton, 1978; Ohlsson & Langley, 1985; Young & O'Shea, 1981) is that it can analyze the child's "scratch" work. Such a system has not been developed before, but the need for this kind of system has been discussed elsewhere (Orey, 1987; Orey & Miller, 1988; Wenger, 1987). Another advantage for this system is that it was developed on a microcomputer (an Apple Macintosh Plus). Because of this, POSIT has the potential of being more available to school systems.

POSIT is designed on the basis of the ACT* learning model described by Anderson (1982, 1987). This model supports, among other things, the notion that the most effective teaching strategy is tutoring. The important aspects of this tutoring strategy are to provide immediate and appropriate feedback as the child is solving a problem in the context of that solution.

The examination of the process the child is using to solve a problem has its roots in a current trend in psychology. As Ericsson and Simon (1984) describe it, "(a)fter a long period of time during which stimulus-response relations were at the focus of attention, research in psychology is now seeking to understand in detail the mechanisms and internal structure of cognitive processes that produce these relations" (p. 1). Many of the earlier diagnostic systems to be discussed also focussed attention on processes. That is, these systems (Brown & Burton, 1978; Ohlsson & Langley, 1985; Young & O'Shea, 1981) attempted to infer the child's process for incorrectly calculating a subtraction problem and described these faulty processes as error patterns. One problem is that these systems only used answers to subtraction problems (or the product of the process) for analysis. This yielded only a fifty percent correct diagnosis rate. In an attempt to increase the likelihood of a correct diagnosis, POSIT examines the student's solution *process* in *process*. POSIT also differs from other systems in that it was designed with a diverse student population in mind. I begin this description with a short review of the problems facing our schools today.

In a recent study, Will (1986) reported that 1.8 million children (4 percent) in our public schools were classified as "learning disabled" (LD). Furthermore, she estimates that there are another 10 to 20 percent who are not classified, but do have learning problems. These children (as many as 10.1 million, classified and unclassified) with learning problems can mature to be high-school drop outs, illiterate, unemployed and/or employed adults living below the poverty level (Will, 1986).

The problems of educating these children are many (Will, 1986). First, the 10 to 20 percent of children who have not been classified are not being helped by special programs. These children tend to fall "through the cracks." They have two paths to failure. They can remain outside of the special programs until they fall so far behind that they can qualify for the programs. Or, if they take long enough to reach this level, they can drop out.

A second problem with our present system for educating children with learning problems is related to the idea of removing the child from the regular classroom and isolating them in special classes. These special classes have a stigma that adversely effects these children's achievement, these include "...low expectation for success, failure to persist on tasks, the belief that failures are caused by personal inadequacies, and a continued failure to learn effectively" (p. 7, Will, 1987). But, keeping the child in the regular classroom has its own set of problems. For example, elementary teachers (K-6) are often (48 percent) unqualified to help handicapped children (Weiss, 1987).

Will (1986) does not "paint this bleak picture" without suggesting some general solutions. For the most part, these solutions depend on the institution adapting to the child's needs. Two of these resolutions directly impact on the purpose of the present discussion. They both deal with time. Children with special needs require more instructional time. The more time spent on a task, the more successful the child will be. This "time on task"[1] must be quality instructional time such as that provided by a tutor. Also, the regular teacher needs to have time for preparation and collaboration with the special programs teacher. Because many classroom teachers are unqualified to help handicapped children, they need to spend time with the school's special education teacher to learn what instructional strategies will be appropriate for the children in their classes. While children are working on computers, the teacher is free to interact with other teachers and improve regular instruction.

Computer technology may help to solve both of these problems. Using a computer in the regular classroom can permit the child with learning problems to have more time-on-task. Furthermore, with increased computer usage, the teacher will have more time for individualization and collaboration with the special education teachers. There are, of course, some problems with this solution.

Weiss (1987) reports that nearly half of the students (K-6) do not have, or have difficulty accessing computers. Only 25 percent have computers readily available. Compounding this problem is the fact that nearly half of the teachers (K-6) have had little or no training in the instructional uses of computers. These results suggest that if computers are to be a solution, then their potential is severely limited by access and qualified personnel.

Becker (1987) concurs with the lack of hardware in our elementary schools. He estimates that the typical elementary school in 1985 had a total of 6 computers. He found that the major role of the computer in the elementary school is for enrichment and to a lesser extent, remediation. For the present discussion, remediation is of interest. Becker reports that the use of the computer by low-ability students is focussed on basic skills. The reported result of working with remediation or drill-and-practice software is "improved motivation, self-confidence, and self-discipline" (p. 156).

Torgesen and Young (1983) suggest that software oriented toward the development of basic skills may be the most appropriate approach for LD children. Children that are facile with these skills can relegate more working memory to higher level problem-solving tasks. Furthermore, they suggest that computers should only be used for instructional tasks that <u>cannot</u> be done with less expensive methods.

Goldman and Pellegrino (1987) point out that mastery of basic skills is indeed an important application of computers in the schools. In this vein, they emphasize both speed and accuracy of the skills. They suggest that through "extended practice" a level of "automaticity" can be achieved. Citing Salomon and Gardner (1986), they review six guiding principles for the instructional software that is intended to develop automaticity of basic skills of its users: 1) Computers are ideally suited for drill and practice; 2) Computers can provide immediate feedback, thereby limiting the practice of incorrect procedures; 3) Over many trials, accuracy improves and speed will increase; 4) Performance is asymptotic, but asymptotes are defined by the individual's ability; 5) Gaming environments may be distracting for learning handicapped students; and, 6) Transfer is a very important measure of skill mastery; a benchmark may be to test transfer between the computer and paper and pencil performance. These principles represent the focus of much of the practice in the schools (more than 50 percent of the computer time is devoted to drill-and-practice (Becker, 1987)).

The basic skill of interest to the present discussion is that of multi-digit whole-number subtraction. Interpreting data from the Fourth NAEP Assessment of Mathematics, Kouba, et al., (1988) found that third graders (approximately 8 years old) only got 50 percent of 3 digit subtraction problems (with regrouping) correct. Further, with problems that required regrouping and had a zero in the tens place of the minuend, these same children only got 45 percent correct. Kouba, et al., (1988) attribute many of these errors to a lack of place-value knowledge. Perhaps a valuable approach for instruction on a computational algorithm, would provide place-value instruction in the context of errors made when using the algorithm. This may be accomplished on an individual basis as the child solves a subtraction problem using a tutoring instructional strategy.

Bloom (1984) described the effectiveness of tutoring in terms of his "2 Sigma Problem." First, "2 Sigma" is in reference to his notion that the average student in a tutoring learning condition performs two standard deviations better than the average student who learns the same topic in a conventional classroom. In addition to learning (achievement), time on task increases and correlations between previous aptitude and achievement test scores and the summary achievement score based on the lesson were reduced. The "Problem" is finding a cost-effective alternative instructional strategy that comes close to the "2 Sigma" effectiveness of tutoring.

Anderson and his colleagues (Anderson, Boyle & Reiser, 1985; Anderson & Reiser, 1985) have pursued this problem in terms of using an ITS. In the case of ITS's, feedback is not only immediate, but the feedback supplies information, in context, to help the learner learn. That is, the feedback is appropriate and meaningful. Anderson et al., (1985) evaluated their Lisp[2] Tutor by comparing it to a human tutor group and a non-tutored group. Essentially, all three groups received instructional materials to read on the topics of the lesson and then were given programming assignments to apply what they had learned. Ten students received help (tutoring) on a computer (the Lisp Tutor), ten students received help from qualified human tutors and the remaining ten students were left on their own to complete the assignments. The computer-tutor group took an average of 15 hours to complete the assignments, the human-tutor group took 11.4 hours and the "on-your-own" group took an average of 26.5 hours. It appears, preliminarily, that computer based ITS's are a viable alternative to the human tutor.

One problem is cost. ITS's are slow and memory consumptive (the Lisp Tutor was implemented on a VAX 725 with 3 megabytes devoted to each tutored individual). However, today's microcomputers have become faster and have much more memory, a trend which is bound to continue. Another problem associated with ITS's is that many of them are not interactive. POSIT was intended to address some of these problems.

Previous Work

Artificial intelligence in education is becoming a separate and dynamic area of inquiry. Recent books by Lawler and Yazdani (1987) and Wenger (1987) have helped to present a synthesis of the research that has been done in the area. Furthermore, groups such as the International Conference on AI in Education or the special interest group on AI in Education within the American Association of Artificial Intelligence (AAAI) organization, point toward this emerging field. Lawler and Yazdani point out the two main thrusts within the field, ITS's and learning environments (such as micro worlds in Logo). Of particular concern to this writing is that of ITS's. Most of these systems focus, to varying degrees, on the notion of diagnosing the knowledge that the learner has for the task. Rather than just listing some of the diagnostic systems that have been developed, a presentation of some organized structure of these systems would seem appropriate. Wenger (1987) identifies a hierarchy of three levels of diagnosis: behavioral, epistemic and individual.

Behavioral diagnosis, the first level in the hierarchy described by Wenger (1987), is the analysis of observable behavior and/or behavior that is supplied to the system. It also looks at the unobservable behavior "which is purely an internal process - or at least which the computer cannot observe" (p. 368). This unobservable behavior must be inferred directly from the observable behavior and is the result of internal processes that must have been carried out to get the learner from one observable behavior to the next.

The next level is that of the epistemic. In this level of analysis, the learner is analyzed in terms of his knowledge state. For subtraction, this would be an analysis of both declarative and procedural knowledge that the child has. This is the level of analysis that deals with cognitive structure. Rather than analyzing the behavior the child exhibits and classifying it (such as "bugs" in the DEBUGGY system), the epistemic level is concerned with the knowledge structures that the child has (such as REPAIR theory has done for the DEBUGGY system).

The last level is that of the individual. This level must take into consideration the individual differences of the learner (Wenger lists 8 specific aspects that ought to be considered with respect to individual performance (eg., architectural, learning, stereotypical, etc.)). He concludes that, "though the individual level has not been subjected to on-line diagnosis, assumptions about it have played some role in the design of diagnostic systems, and a view of a student model with three distinct levels will help to make these assumptions more explicit" (p. 371).

Furthermore, Hartley and Sleeman (1973) identified four distinct knowledge bases that all ITS's should have. First, the system should have a fairly complete description of the task environment or domain knowledge. Second, the system should either have a history or have the capability of building a model of the student's knowledge or behavior (this model can be at any of the three levels of diagnosis described by Wenger (1987)). Third, the system should have a teaching component to alleviate lack of knowledge or correct faulty knowledge of the student. The last component ties the others together. That is, a component that determines the most appropriate teaching strategy given the content domain knowledge and the student's model of the content knowledge (or the student's cognitive structure).

Perhaps the best way to describe existing tutoring systems is to describe each system in terms of the epistemic model described in Wenger (1987). Of particular interest are the systems described under the heading of "direct credit/blame assignments" (see Figure 1). Those are Johnson's (1985) PROUST, Burton and Brown's (1982) WEST, Ohlsson and Langley's (1985) DPF (or as Wenger cites it, ACM's path finder), Brown and Burton's (1978) DEBUGGY, Young and O'Shea's (1981) PS^3, and finally, Anderson's (1982) ACT* model of learning applied to tutoring.

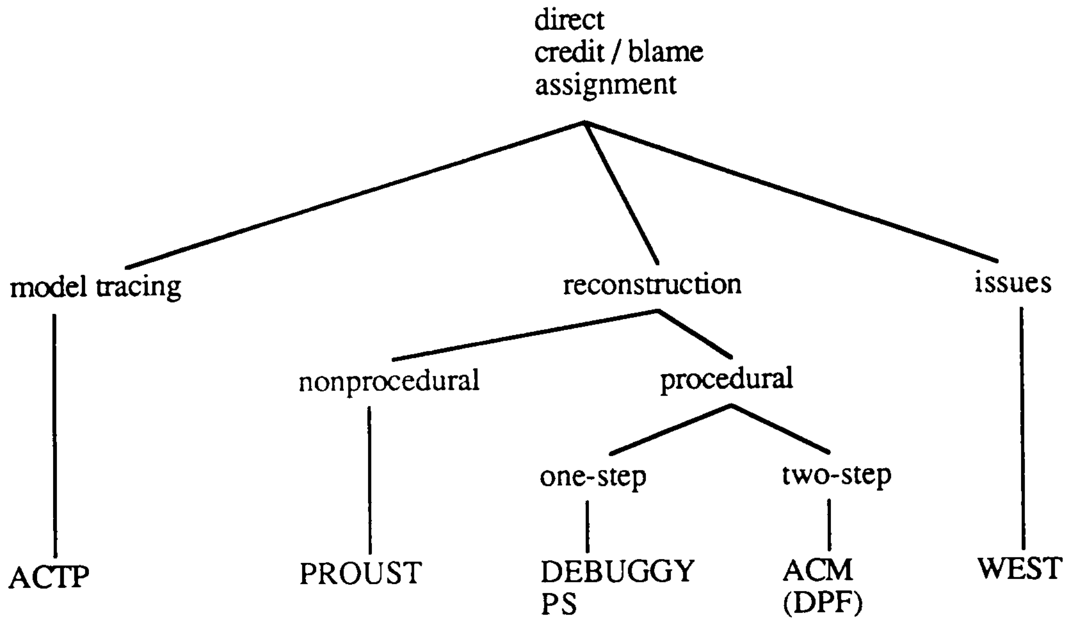


Figure 1. A portion of Wenger's (1987) epistemic diagnosis chart (p. 377).

The reconstructive branch of Wenger's diagram (Figure 1) refers to the programs that attempt to reconstruct the intentions and the strategies for bringing about the intention. In terms of PROUST (a non-procedural diagnostic system), the program needs to reconstruct the student's intended solution to a specific Pascal programming assignment. To do this, the PROUST system is provided with a general framework for the solution of the problem. This is a very general framework. For example, a common first program for students to write when learning to program is to simulate a checkbook. A general framework would be that there needs to be input of the form of checks and deposits. In addition, there would have to be a variable to keep track of the balance and that checks would have to reduce this balance and deposits would have to increase this balance, and so on.

Given this general framework, PROUST then examines the students Pascal code to determine where the student's program is flawed. This is not a general analysis of syntax, (which compilers have done for years), PROUST examines the students Pascal code for semantic flaws. Incorporated into the program is a set of common errors (bugs) that help pinpoint these flaws. The approach taken by this system is to reconstruct how the student is attempting to solve the problem (such as the checkbook simulation). After determining the strategy that the student has taken, PROUST maps (or overlays) the student's solution structure to the framework initially provided and attempts to determine discrepancies.

There are two major strengths inherent in Johnson's approach that could be used in a different content domain such as subtraction. The first issue is that PROUST tries to determine the intentions that the student had for the solution that was attempted. That is, there are a multitude of strategies that could be taken to solve the checkbook problem. In fact, it is quite common that the code of each student in the class would be different. This "openness"[4] is particularly important for helping students with individual differences. PROUST attempts to solve the problem of individual differences by analyzing student solutions in terms of their intentions.

The second point, which I have made before (Orey, 1987; Orey & Miller, 1988), is that a Pascal program can be viewed as a highly specialized problem-solving language. That is, the actual code of the program serves as a sort of protocol of the student's solution to the problem. The problem with applying this approach to subtraction is that it is reconstructive; it attempts to reconstruct intentions after the student has completed a solution. A more direct and appropriate approach for a procedural diagnostic system is to diagnose problems while the solution is being generated.

Wenger (1987) classifies WEST as an issues-oriented system (see Figure 1). It is not (like PROUST, DEBUGGY, PS and DPF) a reconstructive system. WEST is a gaming program that was developed by Burton and Brown (1982) and designed for instruction on operator precedence. This program, unlike the other diagnostic systems to be reviewed, focused considerable attention on pedagogical issues, or "coaching." This game was basically designed to have students move from the town at point 0 to home at point 70 by applying addition, subtraction, multiplication, division and parenthesization on three random numbers. There are certain strategies that are built into the game (such as short cuts, landing on a town allows you to move to the next town, and the ability to bump your opponent back two towns if she is not in a town). The diagnostic part of this game pits the players moves (calculations) against a model expert's.

There are two important issues that Burton and Brown raise with regard to intervention (instruction or coaching). That is, *when to* intervene and *what to* say once you have interrupted play. In order to resolve these issues, they list twelve principles that should be followed when designing the instructional aspect of an ITS (e.g., do not intervene twice in a row, give both positive and negative feedback, automatically adjust the level of difficulty if the learner consistently fails, etc.).

WEST also employs a reconstructive process for the diagnosis of strategies used by the player. This is done by keeping track of the strategies that the player has made in each move. These strategies include information such as direction, magnitude, proper use of the order operations and consideration of the game's built-in strategies (like short cut, town or bump). At the same time that the diagnostic system is evaluating the player's behavior, the "coach" is keeping track of optimal moves that the player has not selected or encountered so that it can make a decision as to whether intervention is necessary. Intervention is necessary when there is a strategy that the player consistently does not attempt and the coach can point out. (Actually Burton and Brown (1982) report that children playing WEST actually "engaged in a meta-game of 'psyching out' the Coach" (p. 98) just to get responses from the coach).

This system is also an "open" system in that it allows players to use their own strategies, but the coach will offer help in developing better strategies. The selection of creative strategies by the player, if they are better than the best the expert can do, would prevent the coach from intervening. However, for all intents and purposes, this system is "open".

The problem with this system in terms of LD students is the fact that it is a gaming environment. As Goldman and Pellegrino (1987) point out, the goals of the game may interfere with the goals of the instructional content. In terms of WEST, the learning of operator precedence may be difficult when the child need also learn the strategies of the game (eg., short cuts).

The next set of systems are not so "open" as PROUST and WEST. The bug library of DEBUGGY, the rule guided production system of Young and O'Shea and the data-driven nature of DPF (ACM) limit their ability to "recognize" that creative solutions are being employed. As a matter of fact, these systems only classify student errors according to error patterns with little regard for the underlying model of the child's knowledge that may be effecting the erroneous performance. This weakness is highlighted by Orey's (1987) findings that there is little relationship between the knowledge the child has for mathematics and the error pattern that the child exhibits. Further, Rosenberg (1987) says that "an exhaustive enumeration of error types is not a model" (p. 8) because a model[5] must be capable of associating appropriate teaching strategies with the specific error patterns, something that has not been done and I have been hard put to conceptualize.

The pedagogical problem is that stating the error pattern will be of little use to the child. The best that can be done is to explain the error pattern to the child. However, this requires the child to reconstruct her solution strategy and the decisions which she made during the process. This will take time and, more importantly, may not be successful because she may have forgotten what led her to use the incorrect strategy. Either way, this instruction does not lead the child to a better understanding of a correct solution strategy.

Rather, it helps the child more fully understand an incorrect strategy. As will be described shortly, a more efficient and effective teaching strategy is to explain errors immediately following, and in the context of, their occurrence. This will be explained in terms of Anderson's (1987) ACT* model of learning.

Another problem with these systems is that they all analyze a procedure (a subtraction algorithm) in terms of its product (answer). It is not worthwhile to diagnose products (answers) in a process (algorithm). It makes more sense to diagnose, on-line, while the student is following the process (i.e. evaluate a process in process, rather than when it becomes a product).

Let us now turn to these systems under the procedural branch of Wenger's diagram (see Figure 1). These systems are labeled procedural because all of them evaluate procedures. The first of the three systems that I will be discussing is DEBUGGY (which was the first of the three to be developed). DEBUGGY uses an extensive bug library (last I read, there were 110 primitive bugs and 20 complex bugs which could be combined to form 1200 compound bugs (Burton, 1982)). The original 130 bugs were built into a data structure called a semantic network. This network could then be utilized to both generate test items that would reveal the bugs and be used to diagnose children's errors. The matching of the child's response to a bug's generated response (over a set of problems) would yield the classification of the child as performing a particular buggy procedure. This is a "closed" world program because if the error the child is manifesting is not part of this library the program will fail. (As opposed to an "open" system which has strategies for providing appropriate instruction when it can not determine the child's solution strategy.) The program also fails when the child is not being systematic in her solution process.

To a child that is just learning subtraction, the problems posed in column format are just that, problems. To assume that the child is going to be systematic is wrong headed. As Newell and Simon (1972) point out, problem solving requires some creativity on the part of the solver. That is, there is no reason to believe that a child will not employ a trial and error process constrained by certain prior knowledge that the child has for mathematics. In general, children will be systematic, but there is no guarantee. Of the erroneous solution sets supplied to DEBUGGY, DEBUGGY can only classify fifty percent as entirely "systematic". Perhaps, this is because the other fifty percent of the children are using problem-solving strategies on each problem which are not systematic.

Young and O'Shea (1981) attempt to make more of a model of subtraction knowledge than the others. Essentially, Young and O'Shea analyzed the types of errors that children make and attempted to cluster them into general categories. The result of this analysis is that they found six algorithmic errors and three pattern errors. These errors were then written as individual productions in their system. Or, as Anderson (1987) describes the initial specification of a production rule, the errors are coded as declarative knowledge within the structure of knowledge of subtraction.

I should note in passing that Brown and Van Lehn (1980) have made DEBUGGY more meaningful through their theory of REPAIR. So, to use repair theory as a guide to explaining the encoding of these declarative productions within a child's cognitive structure, at some point the child is presented with some conflict (impasse) with his subtraction algorithm and makes a repair to his algorithm to resolve the conflict. In terms of the PS structure, the encoding of the production that *IF* there is a zero in the minuend *THEN* write the digit of the subtrahend in the difference space (0-N=N), is done by the child when he first encounters a problem with a zero in the minuend. The child has an algorithm that does not account for zero's in the minuend and, therefore, must repair his algorithm to account for this problem.

The most recent approach to diagnosis is that of Ohlsson and Langley (1985) who used a particular application of their Automated Cognitive Modeling (ACM) program for the diagnosis of errors in subtraction (Diagnostic Path Finder, DPF). They attempted to circumvent the non-systematic error problem by examining each problem the child solves independently. As a consequence, DPF has difficulty establishing a correct diagnosis for the systematic child. This problem notwithstanding, there are some aspects to the DPF system that have some merit. One of the more impressive attributes is the fact that Ohlsson and Langley utilize psychological constructs to serve as delimiters for their best-first search through the diagnostic paths. This explicit marriage of cognitive psychology and AI is of considerable importance to a learning-theory-based tutoring system. Other important characteristics of the DPF system are the fact that it is a general diagnostic system which they applied to subtraction and that the use of a bug library is unnecessary.

The first characteristic, that DPF is a general diagnostic system, is impressive. However, I doubt if the system could be utilized in less logical domains such as reading and writing. Natural language poses its own set of problems that are not easily studied within the production system framework. DPF produces a production system as a diagnosis and utilizes a production system when specifying the task environment.

The second issue is quite interesting. DPF does not require a bug library, but can instead produce a production system for many of the bugs found in the DEBUGGY system (see Wenger, p. 217). When contemplating the development of a DEBUGGY type system for some other area, say addition, it seems overwhelming to think of all the time it would take to produce the bug library (although many people have tried, such as, Blando, Carlson and Sleeman, 1986 (error of precedence) or Kheong, 1987 (fractions)).

One other positive attribute of the DPF system is the fact that it is more open than the other two error pattern diagnostic systems described. However, the system is product oriented (although it does not have to be) and, as I have said, knowing that a child is following a particular error pattern does not have any clear pedagogical implications.

Another set of programs appear to have more potential in terms of being interactive tutoring systems. These systems were developed from a computer-based learning theory

(Anderson, 1982). Anderson and his colleagues (Anderson, Boyle & Reiser, 1985; Anderson & Reiser, 1985) used this model of human learning to develop tutors for both Lisp and Geometry. Wenger (1987) classifies these programs as "model tracing" as opposed to the "reconstructive" (PROUST, DEBUGGY, PS and DPF) and "issues" (WEST) oriented systems previously discussed. In model tracing, the student's performance is matched against correct and incorrect solutions in order to "understand" the current state of the student's solution process. In order to explain this process, a more detailed account of the design of the Lisp Tutor and the Geometry Tutor is necessary. These systems will be explained in terms of Anderson's (1987) ACT* (which stands for Advanced Computer Tutoring - star) learning theory.

At the fundamental level, Anderson (1982) distinguishes between two types of knowledge: Declarative and procedural. Declarative knowledge is a set of facts about a specific skill which can be verbalized. The initial step in learning a skill is to encode the declarative knowledge about that skill. This verbal factual information can be rehearsed to maintain these facts in working memory while attempting to carry out the skill.

The second step in the acquisition of a new skill is to convert the declarative knowledge into specific rule-based actions or procedural knowledge. This conversion from a verbalizable piece of declarative knowledge (which is stored in a network) to an "automatic" firing of the rule has been termed by Anderson as knowledge compilation. These actions may be actions for acquiring more knowledge or acting on existing knowledge (Goldman & Pellegrino, 1987). In the ACT* theory, procedural knowledge is represented by productions and declarative knowledge is represented in an associative network (Anderson, 1982). At the final stage (after many hours of practice), the procedure speeds up and is highly accurate. Anderson calls this the procedural stage while others would refer to it as the development of automaticity (Fitts, 1964; Goldman & Pellegrino, 1987) or tuning (Rummelhart & Norman , 1978)

How all this gets stored in the ACT* production system is important for the understanding of the tutoring systems. A production consists of two parts; a conditional (or goal state) and an action (to use existing knowledge to achieve the goal or set up sub-goals to acquire the necessary knowledge to achieve the goal). As examples, Anderson (1987) presents two "Englishified" productions from the Lisp Tutor:

P1:	IF	the goal is to write a solution to a problem and there is an example of a solution to a similar problem.
	THEN	set a goal to apply that template to the current case
P2:	IF	the goal is to get the first element of List 1
	THEN	write (CAR LIST1) (p. 193)

In these two examples the two types of actions are represented. P1 (for Production one) is an example of using analogy to solve a problem. In order to solve the present problem, find a similar problem and try to map the previous problem's solution to the present problem. This production represents a process of acquiring more information to solve the problem.

The second example, P2, demonstrates how to use existing knowledge in a production. The goal specified in the conditional part of the production is to get the first element in a list. There is a primitive function in LISP which performs this function: CAR (Contents of the Address Register, if that helps). This goal can be achieved by the application of existing knowledge.

In addition to these correct productions, the Lisp Tutor and the Geometry Tutor have a set of incorrect productions which have been collected from protocols of students in these content areas. Wenger (1987) presents a buggy production taken from Reiser, Anderson and Farrell (1985):

IF	the goal is to combine LIST1 and LIST2 into a single list, and LIST1 and LIST2 are both of the type list.
THEN	Use the function LIST and set subgoals to code LIST1 and LIST2. (p. 294, Wenger, 1987).

This is a common error made by beginning programmers of LISP. If LIST1 is (John is) and LIST2 is (a big cat), you may want to combine these lists to make (John is a big cat). However, by using the LIST function, the result is ((John is) (a big cat)). The correct function is to use APPEND.

In model tracing, the Lisp and Geometry Tutors use a set of correct productions (to model correct solutions) and a set of "mal-rules" or incorrect productions (to model incorrect solutions). Because these tutoring systems are interactive, each step the student takes in a solution to a problem is matched against these productions. Associated with each faulty production is an instruction to correct this faulty production within the learner. If the student's step does not match any of the productions, the program interrupts and gives the next appropriate step in the solution of the problem (Anderson, Boyle & Reiser, 1985). In this sense, this is a "closed" system. The student's solution steps need to match the productions supplied to the system. Anderson and Reiser (1985) reported that there were 325 correct and 475 incorrect productions in the Lisp Tutor. Anderson and Skwarecki (1986, as reported in Wenger, 1987) report that the total number of productions now exceed 1200.

Notwithstanding the problems associated with the "closed" system and the large number of rules, Anderson et al., (1985) suggest some important design characteristics of an ITS. First, as just described, the system needs to be able to model correct and incorrect solution strategies. This is done in these tutoring systems by using correct and incorrect productions. Second, instruction must be given in context. In the case of incorrect productions, when the student's step matches one of these productions, instruction is given to explain why this is a faulty solution step. Third, this instruction must be given immediately. Because of the importance of immediate feedback, an ITS must be interactive. Finally, the user interface for the system must be designed to reduce the load on working memory. In the case of the Lisp Tutor, an editor was created to reduce the amount of syntax that the student must hold in working memory.

With regard to the second and third point, POSIT was designed to be an interactive system. Anderson (1987) describes the importance of immediate feedback in this way:

> The importance of immediate feedback is an interesting consequence of the interaction between compilation and working memory limitations. For knowledge compilation to work correctly with delayed feedback, it is necessary to preserve information about the decision point until information is obtained about the right decision. Then a production can be compiled attaching the correct action to the critical information at the decision point. Working memory failures will cause information about the decision point to be lost and consequently incorrect operators will be formed. Such failures will increase with the delay of the feedback (p. 203).

Summary

Three systems were discussed which were designed in the tradition of error-pattern diagnosis (Brown & Burton, 1978; Ohlsson & Langley, 1985; Young & O'Shea, 1981). A major problem with these reconstructive systems is that they analyze the child's answer (product) in order to reconstruct their solution strategy (process). Further, this reconstructive approach is inaccurate (accurate diagnosis is at about 50 percent). Finally, because these systems are not interactive, pedagogical issues such as immediate feedback are impossible.

Johnson's (1985) PROUST was presented as an example of a reconstructive system which is more accurate. The reason for the increased accuracy is that a Pascal program is a rich problem solving representation of the solution process. Further, the pedagogical implication is that instruction can be given in context (although not immediately), assuming the learners can also reconstruct their solutions given their code. This is, again, due to the fact that the Pascal code is a high level representation of the solution process.

Immediate feedback can be given in context only in an interactive system. WEST (Burton & Brown, 1982) is an issues-oriented interactive system. As a result, Burton and

Brown make some suggestions as to when to interrupt a student and what to say once the student has been interrupted. A problem associated with this system is that it is a gaming environment. This may not be appropriate for student's with learning problems (Goldman & Pellegrino, 1987).

Finally, the tutoring systems developed by Anderson and his colleagues (Anderson, et al., 1985; Anderson and Reiser, 1985) were presented because they solve these problems. However, there are some problems associated with these systems. They require a large number of productions for the problem solving domains of Lisp programming and Geometry proofs. Further, this model-based system creates a "closed" world for the learner. Another problem is that the large number of rules required by these systems force the development of the system on computers which are fast and have large amounts of memory. This makes microcomputer implementation prohibitive.

Design of POSIT

The purpose of this dissertation was to design, develop and evaluate a microcomputer-based system that tutors students as they solve multi-digit subtraction problems. The first issue in the design of such a system is the fact that a large number of elementary teachers are not well trained in the instructional uses of computers in the classroom (Weiss, 1987). In order to alleviate difficulties that might arise from this problem, POSIT has two major components. One component is the Tutor (to be described later). The other component is called Teach. This component teaches the child how to use POSIT. A further benefit to this approach is that a child who has a computer at home can independently learn how to use POSIT. In order to explain how the Teach component works, it is first necessary to explain the design of the interface which captures the child's solution "process".

The Interface

In order to study the processes that the child is using when completing a subtraction algorithm, it was necessary to design an interface which would provide POSIT with the child's "scratch" work. This interface is depicted in Figure 2. Areas labeled 1 to 4 and 9 and 10 are *regrouping areas*. Areas 9 and 10 do not initially appear on the screen. These areas serve the purpose of providing a location for multiple regrouping. That is, when you need to "regroup from" and "regroup into" that place value column. The areas labeled 5 to 8 are *difference areas*. These areas provide space for the child to enter the difference for that column.

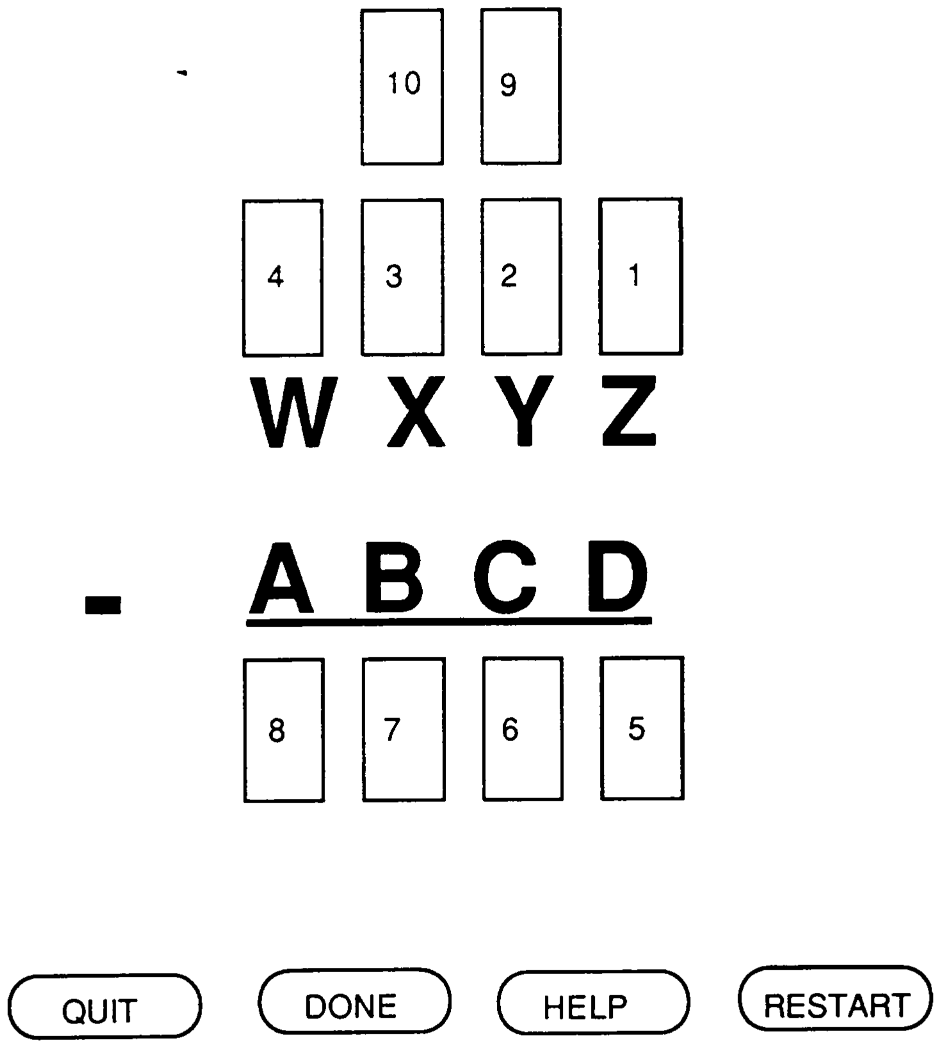


Figure 2. The design of the interface.

The basic approach is to allow the child to point at any area using a computer mouse and to chose that area by pressing the mouse button (clicking). Once an area is clicked, a cursor will appear. At this point, the child can type a number. For example, suppose the child would like to decrement the value in the tens place. The child would click on area 2. This would cause the square to disappear and the cursor to appear. In this case, area 9 would also appear and a line would be drawn through the digit Y (crossing out the previous value). The child would then type the value Y - 1. In this way, all "scratch" work can be monitored by POSIT.

In addition to the subtraction interface, there are 4 on-screen buttons which can be accessed by clicking. The *Restart* button serves the purpose of allowing the child to effectively erase all work and start the same problem over from the beginning. The *Help* button provides the student with one step which draws the solution closer to completion. The *Done* button allows the child to communicate to POSIT that she has completed the problem. Finally, the *Quit* button allows the child to terminate execution at any point in the program. Both the *Quit* and the *Restart* buttons allow the child a second chance. That is, after clicking one of these buttons, the child is asked if he is sure he wants to perform this function.

Teach

In order for the child to use this system, she must be taught the purpose of the squares on the screen, hence the Teach component. This part of POSIT serves two purposes. First, because this is an interactive system, the number of errors that a child makes due to misunderstanding POSIT needs to be minimized. The strategy chosen to teach children to use POSIT was to "walk" them through the system. In this walkthrough, the child is instructed to point the mouse at regrouping and differences areas as they solve an example problem. Then, the child will be instructed to type the specific number which is required. Also, the buttons are explained.

As a consequence of this walkthrough, instruction is given on a subtraction algorithm. Because this is a diagnostic/prescriptive software package for subtraction, the instruction on the use of POSIT must include a step-by-step demonstration of a subtraction algorithm. This part of POSIT is "closed." The child is walked through one specific subtraction algorithm. However, the "open" aspect of POSIT is not limited to one subtraction algorithm.

The specific algorithm that is used for the Teach module is the same as the algorithm used to find the correct states for the Tutor. This algorithm is a parser. The only difference between the two parsers is that, for the Teach module, each function includes a dialogue for the child and an arrow pointing at the area that should be clicked. I now turn to a description of the design of the Tutor component.

Tutor

Get a Problem. The first task for POSIT is to have a subtraction problem for the child to solve. I have decided to allow for two separate approaches for this task. The first approach is to allow the child to enter a problem. In this way, the computer can be used to help the child do her homework or classwork assignments. I have decided to limit the size of the problems that can be used in this system. First, the top number (minuend) must be greater than or equal to 20 (in a problem like 15 - 8, regrouping is redundant). The bottom number must be greater than zero, but, of course, less than the top number. Second, the upper limit of the top number is 9,999. By allowing the top number to be a 4-digit number, the problem of regrouping-across-zeros (an error pattern, see for example, Van Lehn, 1982) can be observed and corrected. However, increasing the size to 5-digit numbers increases the time spent on a particular problem, but does not add much to the learning of multi-digit subtraction (Thompson and Rathmell (1988) suggest that 3-digit problems are sufficient for the development of computational proficiency). Further, the computer system would need to be expanded to handle these extra cases (in the parser, for example) and would make it more difficult to implement such a system on a microcomputer. These constraints need to be checked by POSIT as the child enters an assignment problem. Another constraint in this case is to make sure that the top number is larger than the bottom number.

The alternative to the child entering a problem is having the computer supply one. Appendix A lists 18 random problem generators. Each problem type gets more complex from top to bottom partially based on the Kent State Checklist (it is limited to 3-digit problems) of mathematical knowledge (Underhill, Uprichard & Heddens, 1980). The specific problem complexity is determined by the child. Increases in complexity can only be made by the child after successfully completing a problem. However, if the child consistently makes errors at a specific level, POSIT asks the child if he wants to reduce the complexity. The rationale for making this design decision is based on the notion of learned helplessness (eg., Diener & Dweck, 1978). If the problem complexity were chosen at random or increased after each correct solution (no matter how many prior errors) the child has little or no idea of how she is progressing or even if she is progressing. By giving control to the user, the child can control his own progress. In fact, by reducing the complexity with continued failure, perhaps the child might eventually succeed. A general problem generator cannot control for problem complexity, hence the 18 problem generators.

There is one other control feature for the problem generators. Problems of inference arise with regard to the smaller-from-larger error pattern. For example, suppose that the child enters 5 in the difference area of the ones place. Further, suppose that the digits in the ones place are 1 - 6. The inference problem is whether the child has mentally regrouped (making the response 5 mean 11 - 6) or whether the child used a smaller-from-larger process (1 - 6 is transposed to 6 - 1). Obviously, this could be determined later (when the child operates on the tens place), but since I could control for this, I did. The control is that the difference in the ones place cannot be -5 and differences in the other places cannot be -5 or -4 (controlling for the case of regrouping from).

Getting the Correct Solution States. As Anderson et al., (1985) described, one component of an ITS is to generate the correct states for the problem. In order to accomplish this, a parser was created to generate these states. All possible correct states are listed on the first page of Appendix B. By using the parser, a list of all correct states can be generated for each area. For example, if the problem is 4325 - 2458, then the result of parsing this problem will yield the following list of values:

Area (see, Figure 2)	Value
1	15
2	11, 1, 12
3	12, 2, 13
4	3
5	7
6	6
7	8
8	1

In the case of area 2 and 3, it is necessary to both decrement the value by one and increment the value by ten. Therefore, possible values can be one or the other or both of these operations. Obviously, when only one of the operations is completed in the interaction with the child, the other operation must be carried out in the area above (or can be completed mentally and is determined in the difference area). Thus, after parsing the problem, all possible correct solution states are iterated.

The actual solution strategy the child follows, however, does not need to achieve each of these solution states on the screen. Because many people can complete a subtraction problem using mental calculations, some regrouping states may be achieved and some may not. The only required states are those in the difference areas (the correct answer).

Error States. Anderson et al., (1985) recommend that , in addition to creating correct states, a model of incorrect performance is a necessary part of an ITS. Table 1 lists all of the errors that POSIT examines. Because POSIT examines the values entered in the regrouping areas, a reanalysis of errors was conducted. This reanalysis yielded the four categories of errors shown in Table 1. Two of these error categories were directly drawn from the previous work described earlier; Errors of Creation and Other errors. Errors of

Omission and Commission were derived from a logical analysis of the subtraction process and psychological constructs. First, errors of Omission and Commission will be described.

Table 1. Error categories.

<u>Errors of Omission</u>	<u>Errors of Commission</u>
Increment (add 10)	Increment
Decrement (subtract 1)	Decrement
Both	Both
Neither	
<u>Errors of Creation</u>	<u>Other Errors</u>
smaller-from-larger	Fact error
0-N=N	Typing error
0-N=0	Leading 0
N-N=N	
N-N=0 with regrouping from	
M-S=0, with S>M	

The rationale for Errors of Omission is that working memory capacity is limited (5 to 9 items, Miller, 1956). The child performs a subtraction algorithm in a serial fashion. For example, if the unit's place bottom digit is larger than the top digit, then the child may regroup ten ones to perform the calculation. As the child either searches memory for the correct fact difference or reconstructs the correct fact (eg., counting back), the child may forget to decrement the tens-place digit. This is an Error of Omission.

In the case of Errors of Commission, the child may have had success by decrementing a digit in a specific place value location. The child may then generalize this process (and its success) to always decrementing in that location. This is an Error of Commission, using a sub-process of an algorithm at an inappropriate place. These two types of errors are based on the logical structure of mathematics. However, when a child initially learns subtraction, a subtraction task is a problem. As noted earlier, Newell and

Simon (1972) suggest that creative strategies might be employed to arrive at a solution. This is similar to Repair theory (Brown & Van Lehn, 1980); the child is confronted with a situation which he cannot resolve using his present knowledge. In order to resolve this situation, the child makes a repair to his algorithm. This constitutes a third type of error; Errors of Creation. Many error patterns can be explained in terms of the logical Errors of Omission and Commission when analyzed in process, as POSIT does. However, there are some errors which cannot be explained by these logical rules. For example, one of the most common error patterns is the smaller-from-larger error pattern (Van Lehn, 1982). Other Errors of Creation are adding instead of subtracting and writing a zero as the difference when regrouping is necessary (Young & O'Shea, 1981).

The last category of errors is labeled Other. Sometimes the error is a fact error. Out of 80 problems, Orey (1987) found 6 fact errors (7.5 percent). Out of 1549 problems, Young and O'Shea found 127 fact errors (8.2 percent). Even the most skilled user of a subtraction algorithm makes fact errors. In POSIT, many of these errors can be determined and the correct fact can be pointed out. This process is described in the following section. Also, because POSIT is interactive, the child may make typing errors. These types of errors do not fit into one of the three other categories, but they do occur.

Determining Error. The essence of a tutor is to be able to determine the error that a child makes and provide immediate and corrective feedback. In order to determine the error a child used, POSIT uses three pieces of information. Two of these data points have just been described, correct and incorrect models of solution states. The third data point is called implications and these implications are listed in Appendix B. The list of implications at any given point in the solution process represents those steps that have yet to be completed given what the child has already completed on the task. Therefore, POSIT has the ability of "knowing" what the child should do (correct model), what the child may do (incorrect model) and what the child should do given what the child has already done (implication model). Further, the list of implications represents those items in working memory at any given point. Failure to resolve an implication is assumed to be the result of forgetting.

Given these data points, the process which is used by POSIT is that of a search. How this search is constrained depends on the location of the error. There are three distinct types of errors based on location. First, clicking a regrouping area when regrouping is unnecessary (this is an Error of Omission). Second, typing an incorrect value in a regrouping area. And, third, typing an incorrect value in a difference area.

The first location error type, clicking a regrouping area unnecessarily, can be quickly explained in terms of the problem. That is, the declarative knowledge that the top number is greater than or equal to the bottom number and that the column to the right does not need to regroup anything. It should be noted that the approach for error correction follows the ACT* learning model (Anderson, 1987). That is, procedural knowledge is a compilation of declarative knowledge. Therefore, the error messages are stated in a declarative form (production rule) which have three components. The first component of each error message states the current condition (the IF part of a production rule). The second part of each error statement states the action the child performed. Finally, the third part describes the correct action (the THEN part of a correct production). Using this approach, it is hoped that the child can correct her faulty production by being helped to recognize the condition and learn the appropriate action. Because POSIT is unable to determine if the child has incorrectly interpreted the condition or has an incorrect action, both parts of the production are stated.

The second location error type is when an incorrect value is typed in a regrouping area. This error is determined by the list of correct states determined in the parser. When such an error is made, POSIT checks the error against the error types of Omission, Commission and Creation. If the error is determined, appropriate feedback is given. If the error is not determined, outstanding implications for the area are checked. If there are any, this value is provided to the child, along with an explanation. If there are no outstanding implications, the correct value, from the correct states list, is provided with an explanation.

The third and final location error type occurs in the difference area. The first thing POSIT checks is the corresponding regrouping area. If it has the correct value (or no value when regrouping is unnecessary), the inference is that a fact error has occurred. If there is a value in the regrouping area, the obvious next step is to check if there is an unresolved implication (did the child forget to increment or decrement). The only other possibility (because the regrouping area would have to be correct based on a prior step) is that the regrouping area does not have a value. In this case, the child's response must be checked against the error types. If there is a match, then the appropriate feedback is given. If there is not a match, there are two possibilities for resolving the error.

The first approach is to check if the present column is the right most unresolved column. If it is, then the list of correct states can be used to provide the correct solution information. If it is not, then the second approach is to assume the child is getting ahead of herself. Therefore, the child will be given one step for resolving the right-most unresolved column.

Providing Help. According to Repair theory (Brown & Van Lehn, 1980), at times, situations arise in the solution of a subtraction problem where the child is unable to use his existing knowledge to resolve the situation. To assist the child in making these repairs, a help function is built into POSIT. The help function works similarly to the error detection process just described. One difference is that it never provides more than one step in the solution process to the child. Whereas, the error detection process will provide the child with all the necessary information to resolve a column when an error occurs in a difference area.

The hierarchy for determining the one step to provide the student proceeds like this.

First, the help function checks to see if there are any outstanding implications. If there are, resolution of the right most implication is provided. The second level of the hierarchy checks to see if the right-most unresolved column can be directly resolved. That is, if the regrouping has been completed or is unnecessary, the system requires the child to carry out the operation on that column. The last level of the hierarchy is to provide the child with one step in the regrouping process in the right most unresolved column.

Saving the Results. There is a variety of information that was saved which is more of interest to research than to tutoring. Those data which were saved are the problem, the sequence of inputs (clicks and values, along with messages), time intervals between inputs, and the date and time.

The problem, the sequence of inputs, and the interval of time between inputs actually give more information than "scratch" work. With scratch work, you can see the problem and the actions performed on the problem. However, you cannot determine the order of those actions or how long it took between those actions. Therefore, POSIT provides a relatively richer source of information about a child's solution strategy.

The sequence of inputs includes mouse clicks, value typing, help and restart access, and error messages. This information was used for two evaluation purposes. First, the help and error messages were evaluated to determine their validity. Over a period of time, the system can be improved by providing the system with the necessary knowledge to handle the errors that went undetected. The second purpose is in terms of evaluating learning by the child. Reductions in the number of help, restart and error messages, imply that skill acquisition has occurred in terms of accuracy. The time interval data was used to determine improvements in speed. Increases in speed and decreases in errors imply that the child is reaching the procedural stage (Anderson, 1982) or automaticity is being developed (Fitts, 1964; Goldman & Pellegrino, 1987).

Summary

The major contribution of the POSIT system is that it provides an interface for studying children's subtraction performance while they are in the process of solving a problem rather than trying to reconstruct that process given their answer (product) to a problem (eg., Brown & Burton, 1978; Ohlsson & Langley, 1985; Young & O'Shea, 1981). A deviation from previous on-line tutoring systems is that it provides a component to teach children to use the system. POSIT also deviates from other systems, in that it will be implemented on a microcomputer.

One final point should be made with regard to the content of POSIT. There is a trend within the mathematics education community toward a de-emphasis of algorithmic learning (after all, a calculator can perform subtraction quickly and accurately). If algorithms are to be taught, the emphasis should be placed on using the instruction as a means for teaching mathematical concepts. As Kouba, et al., (1988) point out, many of the errors that children make can be associated with misconceptions of place value knowledge (eg., misunderstanding the place holder 0). Although POSIT provides declarative knowledge at the point where the child makes an error, this information does *not* parallel the conceptual orientation advocated by the current curriculum standards (National Council of Teachers of Mathematics, 1989). This is not to imply that future versions of POSIT could not use an instructional strategy more in line with this orientation.

Method of Evaluation

There are two forms of evaluation which were used in the development of this system: Formative and field testing (Gagne & Briggs, 1979). Formative evaluation takes place during development. As a component of the entire instructional system is developed, that component is "tried out" on a small group of participants. Based on these tests, revisions are made to the component to make it more effective. Field testing, on the other hand, is a test of the entire system (and can be viewed as either formative or summative). Once a complete version of the system is available, a group of participants use the system and improvements are made based on these results.

Formative Evaluation

There are three aspects of the system which were tested during this phase of evaluation: readability of error and help messages, testing the interface in the context of the Teach component and testing the entire system to assure that there were no system problems. Each error statement associated with the error types in Table 1 was evaluated in terms of the child's ability to understand the statement. That is, participants read the statements and said what they thought it meant. An attempt was made to keep the text easily understood.

The second formative test was conducted on the interface. In order to determine if children could easily learn and use the interface, the Teach component of POSIT was developed first. There were two measures of ease of use. First, the number of missed clicks was recorded. Recall that the Teach component points at an area and instructs the child to click that area. If this was a difficult task, the program would have been modified. However, this was not a problem.

The second measure of ease of use was typing errors. Recall that in the Teach module, after the child clicks an area, POSIT instructs them to type the appropriate value. If there are a large number of errors, then a resolution might be to have a separate component which has the child randomly type digits. Also, a numeric key pad may be a viable solution.

Field Testing

As described in the design of POSIT, a variety of data are saved by the system. These data were used to determine if the instructional system was effective. There are three questions in this regard: 1) Is the feedback appropriate?; 2) Do children learn from POSIT?; and, 3) Does the computer based learning transfer to paper and pencil tasks?

In order to determine if feedback is appropriate, the sequence of mouse clicks and numerical inputs was recreated. If the feedback is inappropriate, POSIT could be revised accordingly.

There are a variety of measures to determine if learning has occurred. First, from the initial to the final use of the system (determined from the date and the time), at each level of problem complexity, does the number of errors, help and restart access decrease for each child? Second, holding problem complexity constant, does the time interval between inputs decrease? Each of these analyses were measured in terms of descriptive data: means and standard deviations.

In order to test transfer from POSIT to a paper and pencil task, a pre- to post-test evaluation was conducted using a test generated by DEBUGGY to elicit error patterns (Van Lehn, 1982). Participants were tested before using POSIT. After 5 to 7 hours of interaction with POSIT, a post-test, which is parallel to the pre-test, was administered. Improvements were measured in terms of the total incorrect responses for the test.

Materials

POSIT was developed on an Apple Macintosh Plus with a 20 megabyte hard disk drive and 2 megabytes of RAM memory. Development of the system was done in Allegro CommonLisp by Coral Software. The working version runs on a Macintosh Plus with 2 megabytes of RAM and preferably a hard disk drive. The actual file size is 1.2 megabytes.

As described above, a subtraction test developed by the DEBUGGY system and reported in Van Lehn (1982) was used to test transfer. A parallel form was created by changing the digits in the problems of this test without changing the places where regrouping was required nor the location of zeros. Copies of both of these tests appear in Appendix C.

Participants

Five participants were used in the formative evaluation. These participants were about 8 years old. They were selected based on their parents recommendation as to whether they were poor students of mathematics. For the field test, only those students who were having difficulties in subtraction were used (based on their teacher's recommendation and their performance on the pre-test). Out of a pool of 28 students who were having difficulties in subtraction, twelve participants were selected.

Method

For the formative evaluation, participants were asked to use the system in a university lab environment or at home. In addition, the participants were tape recorded so that the readability of the text could be determined.

For the field test, participants took the pre-test prior to beginning the experiment.

For five weeks, the participants spent 30 minutes per day working with POSIT. After the instruction, participants were administered the parallel form of the Van Lehn test. Also, the participants were asked to do one problem from each of the 18 problem levels on the first and last days of the field test. These data were used to determine if increases in speed and decreases in errors had occurred holding the problem complexity constant.

Procedure

After the twelve participants for the field test were selected, the five weeks of treatment began. The content of the instructional treatment was as follows. First, the participants were asked to use the Teach component. The participants spent one session (30 minutes) on this component. On the second day, the participants were asked to complete one problem from each of the 18 problem levels (Appendix A). This took two to three days to complete. The participants who completed the 18 problem levels before the others were allowed to freely use POSIT in any manner they wished. After everyone was finished with the first task, a chart was created to indicate at which level the child had made at least one error. Each participant was then asked to complete two problems in a row without error at each of the levels where they had made an error in the first part. This gave them practice with feedback on problems where they had difficulty. This took two to three days.

A second chart was created based on the previous task. That is, if the child was unable to solve two problems in a row at each of the previous levels the first time she tried, they were asked to attempt that level again. This took one to two days. At this point, many of the children "accidentally" chose to enter their own problem. Therefore, 17 problems were copied out of an elementary textbook (Payne, et. al., 1985). The participants were asked to enter these problems and solve them. This took about two days. The next task the participants were asked to complete was to solve two problems in a row at each of the 18 problem levels. This took three to four days. Finally, the participants were asked to complete one problem from each of 8 problem levels to compare to the first few days of the treatment.

The treatment lasted five weeks, but, because there were vacations and achievement testing that conflicted with the treatment, the actual number of days of the field test was 17 days. One other comment that should be made about the procedure of the treatment regards the role of the experimenter. In all but two of the 17 days of the treatment, the experimenter attended the interactions with POSIT. The experimenter served two roles during these sessions. First, system errors occasionally occurred during the interactions. The experimenter corrected the problem and allowed the participant to continue. Also, at times, a participant would not understand the textual error message. The experimenter explained the meaning of the message, but did not elaborate beyond the meaning of the message.

Results and Discussion of the Formative Evaluation

As noted in the method section, there were two stages of formative evaluation. In the first stage, the Teach component was used to test the readability of messages and the ease of use of the interface. In the second stage, the entire system was used to test POSIT as it was being used by a learner. In this stage, I was checking to make sure that the problems that were encountered by the participants in the first stage had been alleviated by the changes to the system. Again, readability, clicking errors and typing errors were also evaluated.

Testing the Teach Component

Two participants were used for the first stage of formative evaluation which was conducted in the middle of the school year (December). The first participant, Sam, was in Kindergarten and was 6:1 (years:months). His father characterized him as a good reader, but was unsure of his mathematical abilities. His sister, Jean, was in third grade and was 8:6. Her father characterized her as above average in all subjects except mathematics, where he felt she was about average. In general, neither Sam nor Jean had a great deal of difficulty with the Teach component. The specific problems and the corresponding changes to POSIT are documented below.

Sam. Sam's psychomotor skills caused some initial difficulties in terms of using the mouse. However, after 30 minutes, he had gained some skill. He did not miss-click areas, but initially, it took quite some time for him to get the mouse pointer in the correct area. In terms of typing ability, Sam only made two errors after using POSIT for a total of 30 minutes.

These typing errors presented an unforeseen problem with the system. There was no instructional component which explained that the "backspace" key is used to erase typing mistakes. When an unexpected value is entered, the system "beeps" and informs the learner that a typing error has occurred. However, Sam did not know how to get rid of the incorrect value. Compounding this problem is the fact that each area is large enough to contain two digits. If a third digit is typed, the first two digits scroll out of view. Therefore, if the correct value for an area is 9 and the child types 10, the system will point to that area and display a message indicating that 9 should be typed. If the child types a 9 after the 0 in 10, what is seen by the user in the area is 9, but the actual value is 109.

To correct this problem, the system was changed. What it now does is to clear the area where the error occurred and display a message that tells the learner what he typed and what the correct value should be. In the case described above, the message would be, "You typed 109. Please type 9 in the area which is being pointed to." Requiring him to type the correct value is based on the fact that this is the Teach component. In the Teach component, the user is expected to learn to click areas and type values.

The next problem Sam had with the Teach component of POSIT deals with reading ability. As was noted earlier, Sam was in Kindergarten. I found it surprising that he could read the messages given his lack of school experience. However, what became apparent was that, although Sam could decode the words in the message, he had difficulty interpreting the meaning of the message. For example, every message has the same three parts. First, the message explains why you should type the specific value in the specific area. Second, the message states "To do this, point at the area under/above the arrow and push the mouse button." Finally, the message instructs the learner to type the specific value.

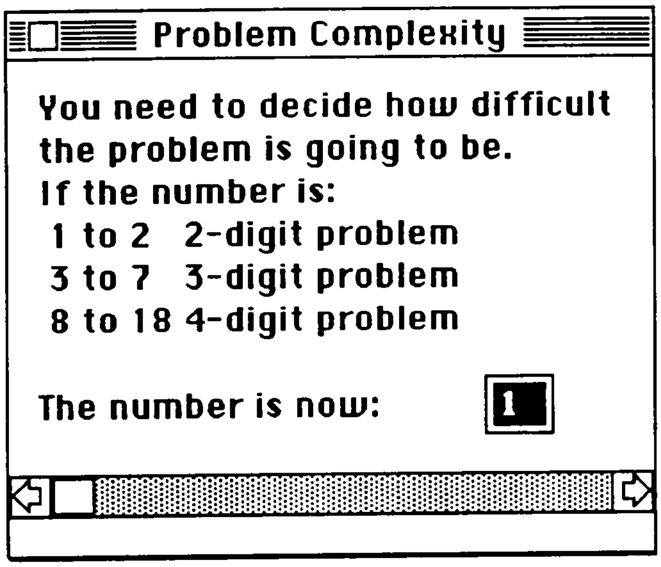
Sam would interpret the second part of the message literally. That is, Sam would place his left index finger in the indicated area on the screen and press the mouse button (rather than using the mouse to point at the area). To alleviate this problem, the second part of the message was changed to "...use the mouse to point at the area..."

Sam's inability to interpret the text in the messages was further demonstrated when asked to explain messages. This inability was related to his lack of knowledge about subtraction and place value. He did not understand words and phrases like regrouping, thousands place, borrow, etc. This lack of understanding did not make him unable to use the Teach component because he was able (after instruction) to follow the directions in the last two parts of each message (click the indicated area and type the specific value). No changes were made based on Sam's inability to interpret the explanation part of the messages. The reason for this is that POSIT assumes that the learner has the prerequisite knowledge of these mathematical terms.

Jean. Jean did not have problems interpreting the messages. She did have some problems with mouse skills, but by the third problem she had, for the most part, mastered its use. However, some other problems arose as she used the system. The least important of these problems is that she used the numbers on the top of the keyboard for input. I pointed out the numeric keypad could be used, but she preferred the numbers at the top of the keyboard. It is not important which numbers are used, but the user should be aware of this option. I pointed out the two options to each participant and some used the keypad while others used the top of the keyboard.

In the Macintosh environment, a typical method of providing a message to the user is to display a window with the message and an *OK* button. All program execution is suspended until the user clicks the *OK* button. In the Teach component of POSIT, error messages and information about the screen buttons (*Quit*, *Restart*, *Done* and *Help*) is displayed using this method. Clicking the *OK* button was not intuitive for Jean. She spent about five minutes trying to perform the next step in the problem before I told her to click the *OK* button. She had difficulty remembering to do this on two later occasions, so I made two changes to POSIT to help alleviate this problem.

First, I changed the text of the *OK* button to *Click Here to Continue*. Second, because other dialogues use the *OK* button in the Macintosh environment, I included text to explain the button the first time it occurs. The first time it occurs is in the window for selecting the problem complexity. Next to the *OK* button in the window this text appears, "Push the *OK* button when you have entered the desired value." In this way, the user can learn to use the *OK* button when it appears in other Macintosh environments.

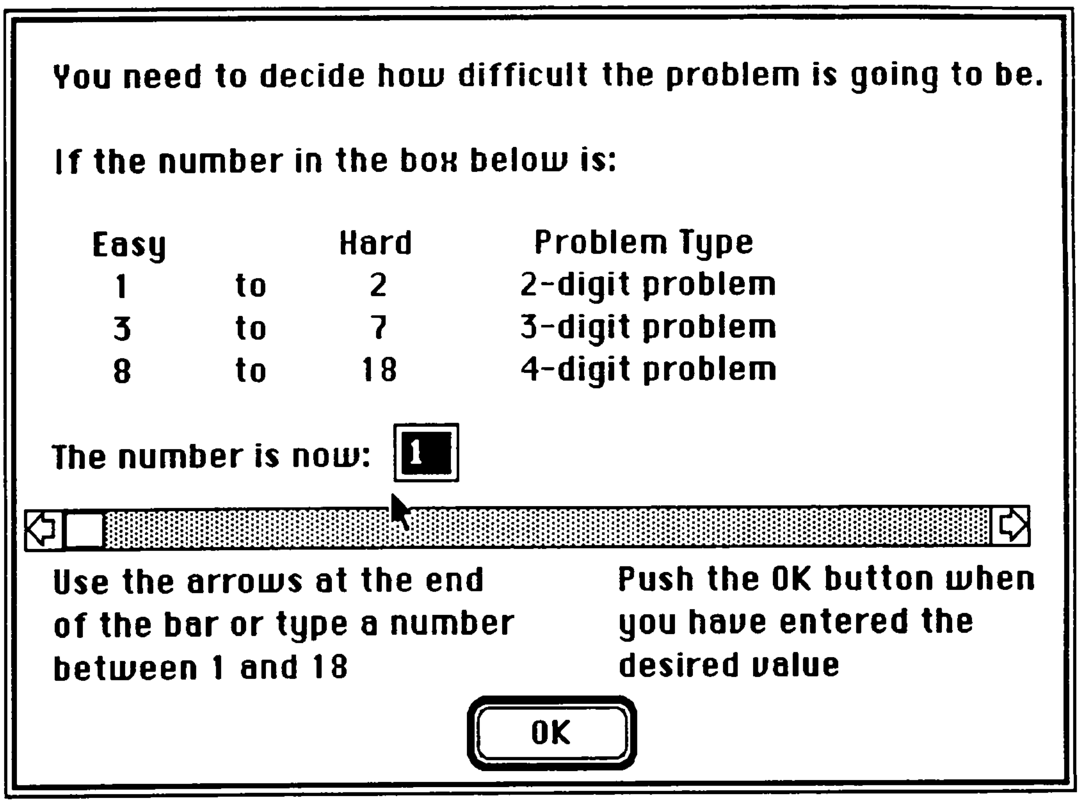

Problem Complexity

You need to decide how difficult
the problem is going to be.
If the number is:
 1 to 2 2-digit problem
 3 to 7 3-digit problem
 8 to 18 4-digit problem

The number is now: 1

Figure 3. The original problem complexity window.

The original problem complexity window appears in Figure 3. There were two problems with this window. First, the close box in the top left corner of the window was difficult to click for Jean and especially so for Sam. Because you cannot change the size of this box, an *OK* button was placed in the window. A button's size can be changed to make it easier to click. As just explained, this button was also used here to explain its purpose in other windows.


You need to decide how difficult the problem is going to be.

If the number in the box below is:

Easy		Hard	Problem Type
1	to	2	2-digit problem
3	to	7	3-digit problem
8	to	18	4-digit problem

The number is now: 1

Use the arrows at the end of the bar or type a number between 1 and 18

Push the OK button when you have entered the desired value

Figure 4. The final form of the problem complexity window.

The second problem with this window was that it was difficult to select a specific value. After about 30 minutes of using POSIT, I asked Jean to select five as the problem complexity. It took her three tries to get that value (clicking the end arrows of the scroll bar three times). Two changes were made to this window to make it easier to select a specific number. First, the length of the scroll bar was increased. By increasing the length of the bar, the speed with which the value changes is decreased. The second change was to allow the user to type the value. Therefore, if I asked Jean to set the level to five, all she has to do now is type five. Some other changes were made to the text to make it easier for the user to understand the meaning of the problem complexity value. The problem complexity window is now as it appears in Figure 4.

As was noted earlier, Jean had no difficulty understanding POSIT's messages nor did she miss-click areas. However, she occasionally typed values in the wrong area. The reason for this is that the cursor (a blinking bar) could be in any area when a message appeared on the screen. On occasion, she would forget to click the area before typing the value. The result was that she typed the correct value in the incorrect area. To correct this problem, two changes were made. The first problem was that the default location of the cursor (or current area) may or may not be the correct area. The change made for this problem was to remove the cursor from the screen between each step (putting the cursor in the correct area would make it unnecessary for the learner to click the area and would not accurately parallel the process in the Tutor). This corrected the problem of the child forgetting to click an area before typing a value. This program modification, however, does not help if the child clicked the wrong area and typed the specified value. To correct this problem, a second modification was done to POSIT. POSIT's Teach component now waits until the correct value is typed in the correct area. Once this is done, POSIT checks all remaining areas for a value. If there is a value, POSIT informs the learner that the *backspace* key can be used to erase typing. This serves two purposes. It checks for typing in other areas and it informs the user of how to erase typing.

One final anecdote should be mentioned before discussing the results of the formative evaluation of the entire system. Sam and Jean are brother and sister. They came over to my house with their father to participate in this phase of the study. After they had finished the tasks that I wanted them to perform, I left them on their own and went upstairs to share a cup of coffee with their father. When we finished the cup of coffee and it was time to go, it was difficult to get Sam and Jean to quit the program and get ready to leave. I mention this because it is not common for a parent to try to convince his children to stop doing subtraction problems.

POSIT - The Whole System

Three participants were used for this phase of the formative evaluation. This phase was conducted in early February. Moe was 11:0 and was in the fifth grade. Moe was having some difficulty in mathematics at the time of the study. His mother characterized his abilities as strong in all subjects, but mathematics. Deb was 8:6 and was in the third grade. Her aunt characterized her abilities across subjects as average. In addition, her aunt noted that she was very shy and might be hesitant about being tape recorded. Pat was 9:0 and was in an advanced mathematics course at the third grade level.

Moe. Moe had no difficulty reading messages or using the interface. He made no typing errors or miss-clicks. In fact, Moe became bored with POSIT because he had already mastered the skill of whole-number subtraction. However, while using the Tutor for about one hour, POSIT found three errors in Moe's solutions. Two of the three were fact errors while the other was an Error of Omission. First, the Error of Omission will be reconstructed.

Moe's error occurred at the point in his solution indicated in Figure 5. It seems that POSIT correctly identified the error as an Error of Omission. The message that appeared was, "It appears that you only added ten to the top number. You also have to complete the borrow into the ones place by taking away one of the tens. So, the correct value for this area is 6. You typed 7. Please type 6 in the area which is being pointed to." Given the information in Figure 5, it seems likely that POSIT correctly identified Moe's error.

However, the error message begins "It appears..." because forgetting to decrement may not be the error he made. For example, he may have mentally decremented and made a fact error (i.e., 13-7=7). This seems unlikely, but there is no way of knowing for sure and telling him it is definitely an Error of Omission may cause confusion.

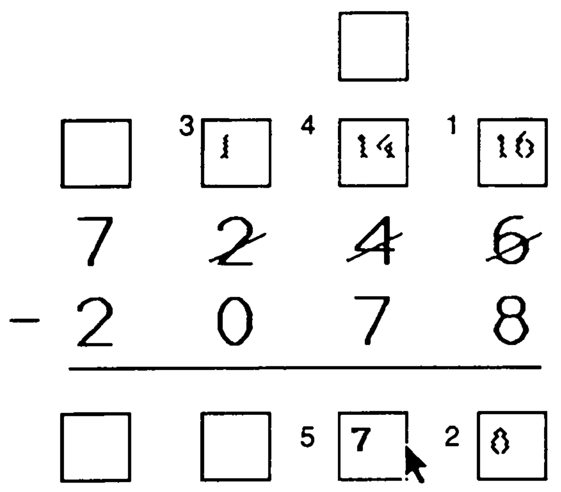


Figure 5. Moe's first error. (Note: The small numbers indicate the order that each step was taken. Also, the numbers in boxes labeled 1 to 4 have been made inactive. Therefore, they appear gray on the screen and unclear here.)

The next two errors that Moe made were determined by POSIT to be fact errors. The first of these errors is depicted in Figure 6. At step 2, POSIT displayed the message, "I believe you made a fact error. The correct value for this area is 0. You typed 4. Please enter the correct value." Because this error is a common error pattern (N-N=N), this error type was added to POSIT's Errors of Creation analysis.

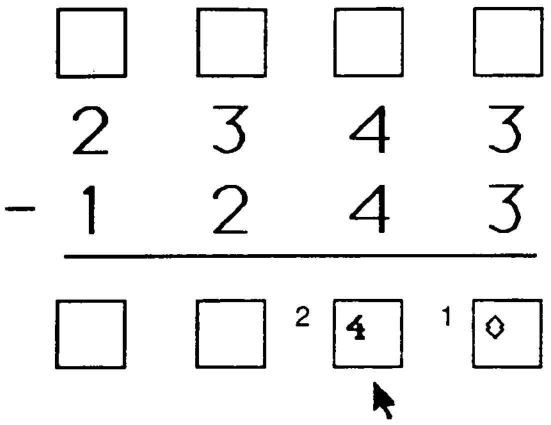


Figure 6. Moe's second error.

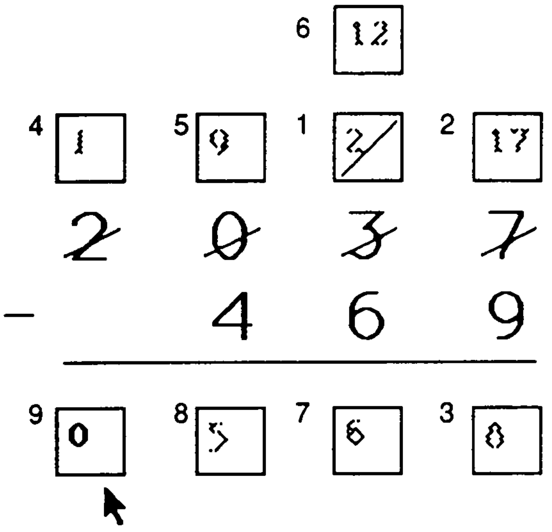


Figure 7. Moe's third error.

The last error detected by POSIT was again classified as a fact error. His solution is depicted in Figure 7. In this case, the error may be related to the pattern 1-Blank=0. There are two reasons for assuming that this is the error. First, two problems before this error, the problem was 9571-384. In this case, Moe correctly determined the value 9-Blank=9. Second, after step 8 in the solution to the problem in Figure 7, Moe clicked the *Done* button. POSIT informed him that he had not finished the problem. He was then forced to type what he thought was a leading zero. The error 1-Blank=0 is an uncommon error. I did not change POSIT because this error occurs so infrequently. The error message displayed by POSIT was apparently not correct, but Moe understood his error (when asked by the researcher). At some point a decision must be made as to whether it is useful to include uncommon errors in the error detection module of an ITS. Wenger (1987) suggests that uncommon errors are not necessary in terms of a cognitive model (in fact, he suggests that even a partial model of the student is quite difficult and a partial model is sufficient for most pedagogical decisions). Because Moe understood his error even though POSIT misinformed him, also implies that in terms of an instructional decision, it is unnecessary to use this error type. If this error occurs again, it will assume that a fact error has occurred.

Deb and Pat. Pat made two typing errors and Deb made one. In addition, both Deb and Pat made some errors which were apparently correctly identified by POSIT. Therefore, POSIT's error detection module did not need to be changed based on their interactions. However, some reading problems did arise during these interactions. These problems are, for the most part, related to word choice. The changes that were made are documented here.

First, as discussed in Jean's section, there was some difficulty understanding the text in the problem complexity window. In fact, as Deb and Pat read the problem complexity window, they skipped over the information on the values associated with problem complexity. The text of the window was restructured so that it appears as in Figure 4.

The remaining changes to the text were based on word choice. As described earlier, I included a function that checks for typing in the incorrect area. The message that appears when POSIT finds this error begins, "If you accidentally..." Both Deb and Pat had trouble with the word "accidentally". This word was changed to the phrase, "did not mean to". Similarly, the message in the Teach module associated with the *Help* button used the word "provides". This word was changed to "gives you". In many of the error and help messages throughout POSIT, the words "digit", "larger" and "operation" appear. These words were confusing for both Deb and Pat. These words were changed to "number", "bigger" and "subtraction", respectively.

I should impart one anecdote before ending my discussion of Deb and Pat's interactions with POSIT. As was noted earlier, Deb was characterized as being very shy. When we began the interaction, I sat next to Deb and looked over her shoulder as she performed the operations on the screen. She progressed rather slowly (eg., it took her twenty minutes to select the problem level that she wanted to use). After she did one problem, I told her that the test was over, but if she would like to continue, she could. I left the room for about 30 minutes. In that period of time she managed to complete seven problems (compared to only one in the first 30 minutes). It would appear that the computer Tutor was less imposing than the human tutor.

One Final Note. The changes described above were changes made because of conceptual/design problems with POSIT. This does not imply that these were the only problems that arose during the formative evaluation. As with most "alpha" versions of computer software, some system errors occurred. That is, errors where the system did not work as it should have or giving the learner incorrect feedback. Such errors, although not documented here, did occur and were corrected. With these system modifications completed, a field test was conducted and the results are described in the next section.

Results and Discussion of the Field Test

As noted in the Method section, there were three questions to be answered based on the results of the field test. These questions were: 1) Is the feedback appropriate?; 2) Do children learn from POSIT?; and, 3) Does the computer based learning transfer back to paper and pencil tasks? These questions serve as the structure of this section. Before I begin an attempt to answer these questions, some general results should be discussed.

Twelve children were selected to participate in this study from a third grade class. One of these participants was absent the first two weeks of the treatment and was removed from any analysis. A second participant attended the first four weeks of the treatment, but moved just before the post-testing began. Therefore, the data generated by this participant was used for answering the first questions, but, because no post-test data were available, he was not used for the other two questions. Further, this means that there were eleven participants for the error analysis and ten participants for the learning analysis.

Given this fact, the mean age of the eleven participants who began the study was 8:11 (Years:Months) with a range from 8:3 to 9:8 (see Table 2). In addition to the age of the participants, it is important to keep in mind how much the participants used POSIT during the treatment. There are two measures of the extent to which the participants used POSIT. First, the time that the child spent using POSIT. The mean time that the participants used POSIT was 5:52 (hours:minutes) with a range from 4:58 to 6:47 (see, Table 2). A second measure of the extent to which the participants used POSIT is the number of problems that each child completed over the five week period. The average number of problems completed by each participant was 141 with a range from 108 to 191 problems completed (see, Table 2).

Table 2. General participant description.

Participant	Gender	Age	Time on POSIT	Problems Completed
P1	M	9:0	6:08	191
P2	F	9:7	6:28	149
P3	F	9:1	5:52	143
P4	M	9:8	5:58	135
P5	M	8:7	5:45	117
P6	F	8:3	5:37	119
P7	M	8:10	5:44	171
P8	F	8:3	6:47	134
P9	M	8:7	5:33	151
P10	M	9:2	5:45	127
P11*	M	8:11	4:58	108

Note: I will use the notation P# to denote Participant # throughout this section.

*Did not participate in post-testing.

In terms of mathematical ability, all eleven participants were considered low-ability students by the teacher. The teacher, whose class was used for this study, had made an arrangement with the teacher in the adjacent classroom. This arrangement was to trade high-ability and low-ability students (in terms of mathematics), during mathematics instruction. The result of this "trade" was that the 28 student's who took the pre-test were the students with the least mathematical ability of 55 students in total. Although none of these students were classified as LD, many of them had failed mathematics during at least one marking period. Given this general description of the participants and the extent to which they used POSIT, I now turn to the three questions.

Is the Feedback Appropriate?

In order to describe the appropriateness of feedback (or the accuracy of diagnoses), a description of some exemplar problems will serve to describe what constitutes a good and a bad message. Moe's first error (Figure 5) shows a "good" message. Although other interpretations could be made (that Moe may not have made an Error of Omission), given the data shown in Figure 5, this was categorized as a correct diagnosis and a "good" message. Moe's second (Figure 6) and third error (Figure 7) were classified as bad messages. POSIT classified these errors as fact errors, when they appear to be N-N=N and N-Blank=0, respectively. These "bad" messages are one category of "bad" message; POSIT classifies the error as one type when it is actually another type. The other category of "bad" message is when POSIT is unable to determine the error. When this occurs, POSIT suggests the next step in the solution rather than providing corrective feedback on the error.

This second category of "bad" message can be classified into three categories: 1) When I reconstruct the answer with data that POSIT saves, I am unable to determine the error; 2) When I reconstruct the answer, I am unable to determine the error, but I had asked the participant at the time of the solution what he had done; and, 3) When I reconstruct the answer, I can determine the error. The following examples illustrate the information that was used to determine "bad" messages. It is left to the reader to agree or disagree with the classifications based on these examples.

First, an example of an error where neither POSIT could determine the error nor I could determine the error, appears in Figure 8. This error was done by P8 on the second day of using the Tutor. I happened to be there when this error occurred and P8 was unsure why she typed 3 in area 3. Perhaps she subtracted 4 - 1, but it is not clear from the data that this is true. However, this type of error rarely occurred and its inclusion in the error library would not contribute to learning (especially since P8 did not know why she typed 3). Regardless, this is an example of a "bad" message where neither POSIT nor I knew what the child did.

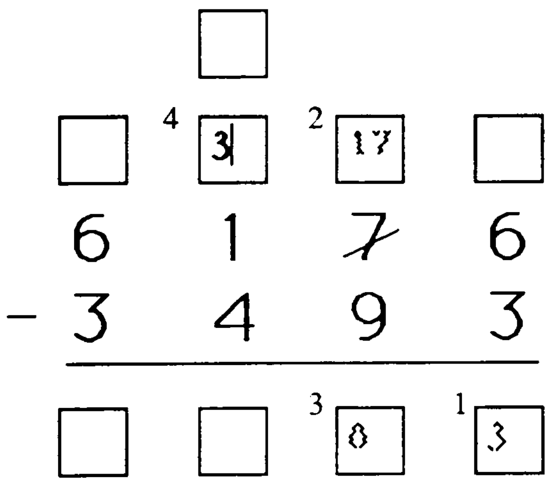


Figure 8. P8's bad message.

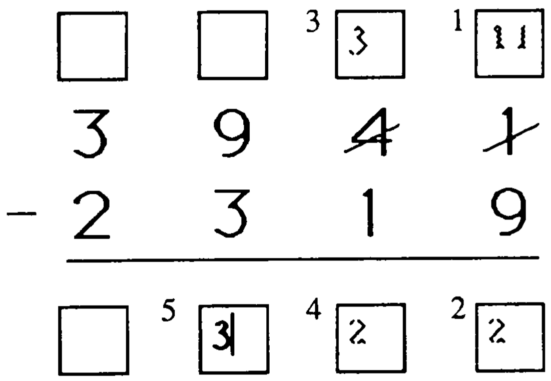


Figure 9. P7's division error.

The second type of "bad" message is when POSIT did not "know" the error and I happened to be there and asked the child what he had done. Figure 9 shows the partial solution of the problem when the error occurred. In this case, P7 found 9 - 3 to be 3 on the second day of using the Tutor. The content of the regular classroom mathematics lessons at this point was learning their multiplication tables. P7 indicated that when he saw 9 and 3 he counted the number of three's in nine. Again, the number of times that this error occurs does not warrant its inclusion in the error library and P7's ability to recognize the error and correct his mistake lends support to this position.

The last type of error where POSIT did not determine what the child did, is when I could determine the error from the data saved by POSIT. P4 solved the problem in Figure 10 to the point where he made the error indicated. This error did occur with some frequency (2%) and would be considered an Error of Creation for the regrouping areas. What P4 apparently did here was to decrement twice. This error might also fit into the Commission category. After all, the first decrement worked, so why not try it again?

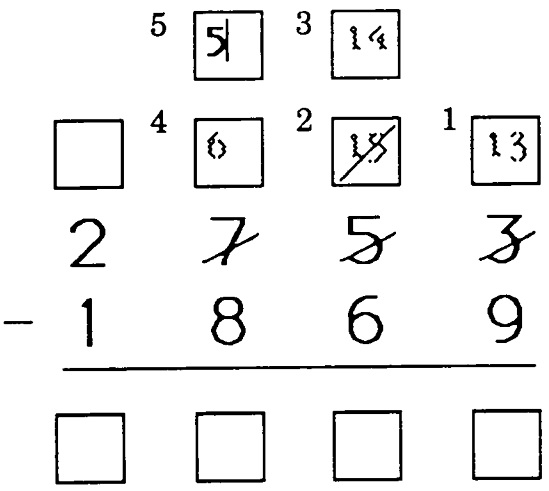


Figure 10. P4's decrement twice error.

Given the above categorizations for determining the appropriateness of error messages, the accuracy of POSIT can now be discussed. Table 3 summarizes the results of the error analysis. The first result of importance is that the current version of POSIT is about 76% accurate. This is far better than other systems (Brown & Burton, 1978; Young

& O'Shea, 1981; Ohlsson & Langley, 1985) whose systems were about 50% accurate. In fact, it is about as accurate as Johnson's (1985) PROUST, 79%.

Table 3. Errors determined by POSIT versus actual errors.

	POSIT		Actual	
Error	# of errors	%	# of errors	%
Omission	180	37	189	39
Commission	72	15	75	15
0 - N = N	2	0	2	0
smaller-from-larger	5	1	5	1
N - N = N	0	0	1	0
N - N = 0 with regrouping	0	0	0	0
0 - N = 0	2	0	3	1
M - S = 0, M < S	0	0	0	0
Fact Error	72	15	58	12
Leading 0	55	11	41	8
Type - 0	29	6	50	10
*No-Clue	72	15	24	5
Other	0	0	41	8
*Identified Incorrectly		9		
Total		489		489

*Excluding the "No-clue" category (which is the classification for when POSIT was unable to determine the error made) and the 9% identified incorrectly, POSIT is 76% accurate.

The second important aspect of Table 3 is the potential for increasing the accuracy of POSIT. In the ideal case, POSIT could be modified and (excluding the 5% "No-Clue" errors in the actual column) be about 95% accurate. Realistically, the accuracy of POSIT could be between 80 and 90%. With three changes to POSIT, the accuracy would have been 89%. Those three changes are described here. First, of the 75 fact errors detected by POSIT, 22 were determined to be bad messages. Of these 22 bad messages, 13 were actually typing errors. These typing errors were the result of typing a character other than a digit. When POSIT converted the input from text to a numerical format, the result was a negative number. It would be very easy to correct this part of POSIT.

The second modification is the result of the fact that 14 of 55 "leading 0" errors were determined to be "bad" messages. Of these 14 "bad" messages, 12 were determined to be Errors of Omission. For example, P9 solved the problem as indicated in Figure 11. He typed the value 1 in area 8 and POSIT classified this error as a leading 0 error because it is unnecessary to type anything in area 8. POSIT can be modified to check that the value was 0 rather than just checking for any value.

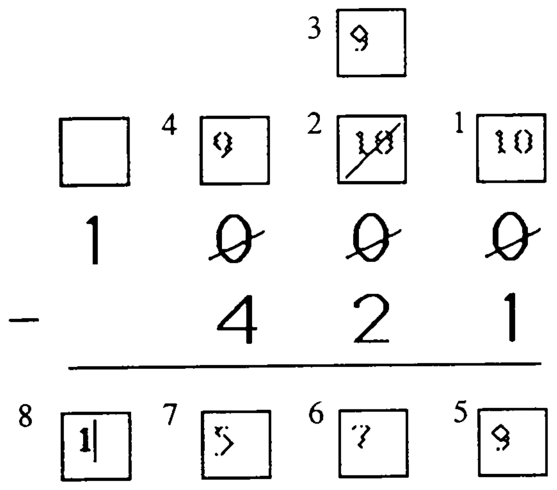


Figure 11. P9's error of omission.

Table 4. Other errors.

Error	# of Errors	%
Add 1	8	2
Decrement twice	9	2
Add 10 to bottom #	10	2
9 for 1	6	1
System Error	7	1
Total	40	

The last change to the system has to do with the category "Other" in Table 3. Of the 41 errors classified as "Other" in Table 3, 40 errors are of the five error types in Table 4. The other error was the same as Moe's third error (Figure 7), N - Blank = 0. This error could also be added to POSIT, but it does not occur very frequently. Of the five errors listed in Table 4, four can be classified as Errors of Creation for the regrouping areas. The fifth error, system error, is an error where an inappropriate help message was displayed and the child was forced to type an incorrect value. This error has already been corrected.

The error "decrement twice" was described in Figure 10. This error occurred 9 times. The error "9 for 1" is a special case of the "decrement twice" error. This error occurs in the middle regrouping areas and is the result of typing 9 when there is a 0 in the top regrouping area or 1 in the top digit. Figure 12 shows P1's version of this error.

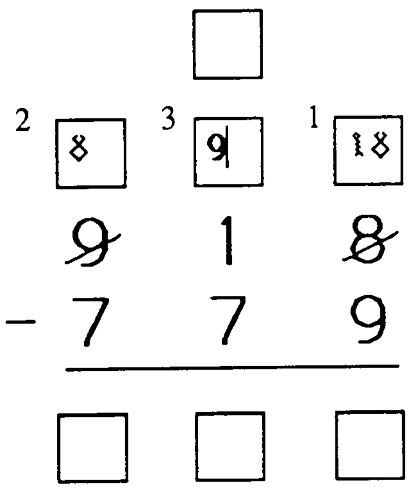


Figure 12. P1's 9 for 1 error.

The other two errors in Table 4 are straightforward. The "add 1" error is when the child adds 1 to the top number instead of decrementing. The other error, "Add 10 to the bottom number," is just that. Instead of incrementing the top number (adding 10), the child adds 10 to the bottom number. The inclusion of these Errors of Creation, as well as the correction to the system, would improve the system by 8% on the data collected during the field test.

The next aspect of the data in Table 3 to be discussed concerns the Errors of Omission and Commission. These errors were clustered together for this analysis, but there are four Errors of Omission and three Errors of Commission (see, Table 1). The way these errors breakdown is shown in Table 5. First, notice that of the 180 Errors of Omission identified by POSIT, 177 were correctly identified. Also, of the 72 Errors of Commission identified, 67 were correctly identified. Second, the most common Error of Omission is forgetting to decrement. This error would correspond to the borrow-do-not-decrement error pattern. VanLehn (1982) reported that Debuggy found this error 12 times out of 157 diagnoses. It is also interesting to note that the other three types of Errors of Omission, only the category "Neither" occurred.

Table 5. Breakdown of Errors of Omission and Commission.

<u>Errors of Omission</u>	<u>%</u>
Decrement	171
Increment	0
Both	0
Neither	6
<u>Errors of Commission</u>	
Decrement	32
Increment	34
Both	1

In the Errors of Commission category, there was a fairly equal split between incrementing (borrow ten when M > = S) and decrementing (taking away one when you did not borrow one). The first type, incrementing, is the same as Young and O'Shea's (1981) production - borrow - when - <. They found that this error occurred quite frequently. The other Error of Commission, decrementing, does not appear in other sources, and yet it occurred quite frequently (32 times or 7% of all errors).

One final comment should be made regarding the errors in Table 3. The most common error is the Errors of Omission, specifically "not decrementing". This was not the most common error reported by VanLehn (1982) and not analyzed by PS (Young & O'Shea, 1981). Further, the most common error reported by VanLehn was the smaller-from-larger error which was the second most common reported by Young and O'Shea. Further still, the other Errors of Creation (which were included based on previous work) rarely occurred. A possible explanation for this difference might be due to POSIT's interface. Because POSIT displays areas above the numbers, the participants may have felt compelled to use them. Since the Errors of Creation are the result of not regrouping, the number of occurrences of these errors was reduced. The fact that there was also a large number of Errors of Commission might support this explanation. Regardless, the frequency of errors differs among the different systems.

Summary

It appears that POSIT is quite successful in its diagnoses. Notwithstanding, the difference in error frequency, the approach to error analysis used by POSIT is potentially very useful. To answer my question, Is the feedback appropriate (or, are the diagnoses accurate)?, I answer, yes. Not only is it more accurate than previous systems, but with some minor modifications it can be even more accurate. I now turn to the question of learning.

Do Children Learn from POSIT?

There were two questions related to learning as the result of using POSIT. The first question was answered by measuring learning directly on the computer. The second question was answered by measuring transfer between the computer and paper and pencil. I will be discussing the results that relate to the first question here; do children learn from POSIT?

There were four measures that attempted to determine if learning had occurred on the computer. These measures can be thought of as pre to post-test measures. That is, as described in the Method section, the students were asked to complete one problem from each of the 18 problem levels at the beginning and again at the end of the field test. A comparison was made between each of the four measures from beginning to end. Those four measures were: 1) The number of times the child clicked the Restart button; 2) The number of times the child clicked the Help button; 3) The average time between each step in the problem at each level; and, 4) The number of errors made by each child at each level. I will begin with the number of times the children accessed Restart.

Recall that the purpose of the Restart button was to allow the child to effectively erase all their work and start the same problem again from the beginning. The rationale for including this measure as a measure of learning was that a child would become familiar enough with the subtraction algorithm after using POSIT for a time that she would not get lost in the middle of the solution and need to Restart the problem. Further, the number of times the children used this button would decrease from beginning to end. As it turns out, this button was rarely used at any point during the study (a total of 11 times out of 1545 problems completed). There was a reduction in the number of times this button was used from beginning to end (two times on the first days versus one time on the last day), but this result is not very enlightening given the infrequent usage. Perhaps children who are first learning the subtraction algorithm would use it more (the children in this study were first introduced to subtraction toward the end of second grade. The field test was conducted toward the end of third grade).

Table 6. Help access from beginning to end.

Participant	Beginning	End
P1	2	0
P2	7	7
P3	2	1
P4	2	5
P5	0	0
P6	20	5
P7	3	1
P8	0	2
P9	0	7
P10	1	1
Total	37	29

The second measure of learning, the number of times the children used the Help button, was somewhat more interesting given that the total number of times this button was used was 198 times. Recall that the Help button served the function of explaining to the child what the next most appropriate step was, given what the child had already completed. The rationale for using this button is based on Brown and VanLehn's (1980) Repair Theory. The idea behind Repair Theory is that the child, at some point in the problem solution, does not know or is unsure what the next step ought to be. That is, their subtraction algorithm is incomplete and they do not know what to do next. The child then creates a Repair to her algorithm to handle the present condition so that she can proceed with the solution. The Help button was included to facilitate correct repairs. The rationale for including the number of times the participant clicked the Help button as a measure of learning, was that as they became more skilled with their subtraction algorithm, they would not feel compelled to use the Help button.

There was an overall reduction in the number of times the participants used the Help button (37 at the beginning versus 29 at the end, see Table 6). However, there were three participants who used the Help button more (P4, P8 and P9), two who used it the same number of times (P2, P10) and one who did not use it either at the beginning or the end (P6, in fact, he only used the Help button once during the entire field test). Of the three participants who increased the number of times they used the Help button, two of them (P8 and P9) did not use the Help button on the first days of the field test. Two possible explanations for this is that they either did not know they were allowed to use the Help button at the beginning or that they did not understand what the Help button did. After the first three days, P9 used the Help button frequently (35 times in total). However, P8 only used the Help button a total of three times.

Table 7. Help access by level from beginning to end.

Level	Beginning	End
1	1	0
2	3	0
3	0	0
4	2	2
5	2	0
6	7	2
7	8	3
8	6	0
9	0	0
10	0	2
11	1	1
12	1	1
13	4	1
14	0	4
15	0	3
16	0	4
17	3	5
18	0	1

The other participant who increased from beginning to end, P4, increased from 2 to 5 times. His overall usage was 25 times, but his usage did not follow a pattern. One day (the eighth day) he used the Help button six times, but the next two days he did not use it at all. In terms of the problem level, P4 used Help at levels 5 and 7 on the first days and 6, 7, 14, 15 and 17 on the last days of the field test. This tendency to use help on the higher level problems is indicated in Table 7. The increase in the use of the Help button occurred at levels 10 and 14 through 18. This pattern suggests that the participants learned the function of the Help button and only used it when necessary (i.e., on the more difficult problems).

Although there was an overall reduction in the number of Help accesses, this measure does not fully support the position that the participants learned to subtract better as the result of having used POSIT. However, the change in where the participants used the Help function does indicate that they learned when it was appropriate and helpful to use the function. The last two measures lend more support for the notion that the participants became more proficient in subtraction.

Table 8. Change in the average amount of time between each step in the solution of a problem from beginning to end.

Participant	Beginning	End (in seconds)
P1	10.7	7.5
P2	14.6	10.4
P3	16.1	8.7
P4	11.4	10.2
P5	18.1	7.0
P6	11.5	8.1
P7	8.9	6.6
P8	23.6	14.6
P9	9.8	9.7
P10	14.6	14.1
Average	13.9	9.7

The rationale for using the pre to post-test comparison on the average time between each step in the problem solution comes from the work done on the development of automaticity (Fitts, 1964), expertise (Rummelhart & Norman, 1978) or procedural knowledge (Anderson, 1982). According to these notions, as a cognitive skill develops, the cognitive process speeds up (a reduction in the time between solution steps on the subtraction algorithm) and is more accurate (a reduction in errors). So, the first measure is speed and, as is shown in Table 8, all participants solved problems faster from beginning to end. However, there was a wide variation in the amount of decrease in time (the best improvement was P5, mean decrease of 11.1 seconds and the least improvement was P9, mean decrease 0.1 seconds) which causes some concern. A further variable that compounds the difficulty in interpreting the results of this portion of the data analysis is the fact that none of the children had ever used a mouse as a computer input device. Therefore, much of the speed up in performance can be accounted for by the development of skill in using a mouse.

Table 9. The number of errors from beginning to end.

Participant	Beginning	End
P1	9	6
P2	25	9
P3	17	6
P4	17	8
P5	9	2
P6	10	1
P7	7	15
P8	19	5
P9	10	2
P10	7	2
Total	130	56

The most direct measure of learning of subtraction in this field test is the measure of errors (see, Table 9). Overall, the participants made 74 fewer errors at the end of the study than they did at the beginning. However, not all the participants improved. P7 actually made eight more errors at the end of the study (15 errors on the 18 problem levels) than at the beginning (7 errors on the 18 problem levels).

Because of P7's performance on the last day, I examined his progress throughout the field test in terms of his errors. Table 10 summarizes his progress and of particular interest is the time he spent on POSIT on the last few days and the corresponding increase in errors. As can be seen on April 3rd, P7 spent 56 minutes on POSIT. On this particular day, P7 spent two sessions on the computer. It appeared to me that P7 became bored with the computer. His teacher suggested that he was not only bored, but as a result, characterized his behavior as careless. The motivating aspects of using a unique computer (the school only had Apple IIe's and only three of these for the entire school) began to wane. To expect a 9 year old child to remain interested in a task for such a long period of time is hopeful at best. Apparently, there was a time during the field test where some improvement might have been found (somewhere before April 3rd).

Table 10. P7's errors, problems completed and time on POSIT throughout the field test.

Date	Errors	Problems	Time on POSIT (in minutes)
3/15	3	16	28
3/16	4	9	18
3/20	5	19	41
3/21	1	11	20
3/22	1	4	17
3/23	0	7	12
3/28	2	12	24
3/31	2	4	19
4/3	9	30	56
4/5	6	11	24
4/6	6	12	25
4/7	6	12	24
4/10	15	24	36

Summary

Taken as a whole, the four measures of learning would indicate that the participants learned from POSIT. The fact that there was a large reduction in the number of errors would support the notion that some of the improvement in times can be accounted for in terms of an improvement in subtraction ability. Further, although there was only a minimal reduction in the number of Help accesses, combined with the above results, the measure would indicate an increased ability to perform subtraction. Given that the participants learned to perform subtraction better, as measured by POSIT (on the computer), I turn to a discussion of transferring this knowledge to a paper and pencil task.

Does the Computer Based Learning Transfer to Paper and Pencil Tasks?

I begin the discussion of transfer with some qualifications as to the results of this section. The pre-test, as described in the method section, was taken from VanLehn (1982). This test was generated by DEBUGGY to elicit error patterns. The post-test was created, again as described in the Method section, based on the pre-test. This test was selected to compare error types from the beginning to the end of the study. Also, the two test forms were used as a one-group pre-test/post-test quasi-experimental design to test for transfer. There are two problems here.

First, because the test being used here is designed to elicit error patterns rather than subtraction achievement, there are no reliability or validity measures for the tests. The second problem is that there are a variety of threats to validity in this experimental design (internal threats -- Maturation, Regression to the Mean, Testing, Instrumentation, etc. as well as the obvious external threats -- the selection of the sample, sample size, etc.). Given the problems associated with the design and the measures, a further problem is associated with the construct of transfer.

Rummelhart and Norman (1978) suggest that you cannot expect transfer to occur until a high level of expertise has been achieved. They suggest that the time for developing this level of expertise varies depending on the complexity of the task, but that it would be around 10,000 hours. Recall that the participants in this study used POSIT between 4:58 and 6:47 (hours:minutes). This is not a long period of time. However, as I have indicated elsewhere, the participants in this study began subtraction the previous year. Therefore, the short period of time in which they used POSIT can be thought of as "tuning" their subtraction knowledge.

Another factor which could influence a person's ability to transfer knowledge from one situation to another is the cognitive distance between tasks. POSIT's interface was very similar (near transfer) to the way the children had learned subtraction. Mayer (1983) distinguishes between two types of transfer. One is directly related the task that was learned (near transfer). The other requires the learner to generalize their learning to other applications, word problems, for example (far transfer). Some people with whom I have spoken with regard to POSIT's interface have discussed the lack of similarity between the regrouping areas and how they regroup (they tend to make little marks next to the top number). However, this was not the case for the participants in this study. Therefore, it is not far fetched to expect transfer from what was learned on the computer to a paper and pencil task.

Given these qualifications, a t-test was run on the data in Table 11. Six participants had fewer errors on the post-test than on the pre-test, 1 made the same number of errors and 3 made one more error on the post-test. As a group, the participants improved from the pre-test ($\overline{X} = 8.5$ errors, $s = 2.4$) to the post-test ($\overline{X} = 5.4$ errors, $s = 3.5$) with a two-tailed probability of $p = 0.037$. Keeping the many qualifications in mind, it would appear that using POSIT had a positive effect on the participant's ability to perform subtraction on a paper and pencil subtraction task.

Table 11. Pre to post-test results of paper and pencil tasks.

	(# errors out of 20 problems)	
Participant	Pre-test	Post-test
P1	13	2
P2	5	6
P3	11	5
P4	10	11
P5	9	5
P6	9	10
P7	8	6
P8	7	7
P9	7	2
P10	6	0
Average	8.5	5.4

In terms of the kinds of errors that were made on the paper and pencil tests, some errors would not occur while using POSIT. One type of these errors is related to spatial orientation. The regrouping areas on the screen while using POSIT are aligned in the column. On the paper and pencil tasks, the participant might write the regrouping value for the hundreds place close to the tens column, for example. When the child carries out the subtraction in the tens place, he misreads the hundreds place regrouping value as the tens place regrouping value. Another type of error that could not occur while using POSIT is related to penmanship. On two occasions, a participant wrote the correct regrouping value of 16. However, the curled part on the bottom of the 6 overlapped onto the top number. When the participant carried out the subtraction, the 16 appeared to be an 11. The last observed type of error which could not occur on POSIT has to do with the carrying process in addition. On several occasions the child borrowed ten, but did not decrement. Also, the top number was equal to the bottom number. For example, if the top and bottom number were 5, then the situation for subtracting was 15-5 (when it should have been 14-5). The "repair" that was made was to determine the correct difference, 10, and write 0 in the difference area and carry the one. POSIT would have interrupted when the child typed 0 in the difference area.

The majority of the errors that were made both on the pre-test and the post-test were Errors of Omission; they forgot to decrement. This result is not at all surprising given that these errors were the most common found by POSIT. This is certainly an explanation for the different frequency of errors compared to VanLehn (1982) and Young and O'Shea (1981).

Participants' Model of POSIT

Wenger (1987) suggests that an ITS should not only model the behavior of the user, but attempts should be made to determine the model developed by the user for the program. POSIT does not perform this function nor was an attempt made to systematically determine this. However, I offer a few anecdotes to explain how the participants came to know POSIT.

The first two of these anecdotes cut directly to the instructional effectiveness of POSIT. These two comments were overhead by me as two participants were learning to use the system. The first comment was, "Just type the number, you don't need to read all that." The word "that" refers to all the explanatory text in an error statement. The second comment was, "Just click Help, you don't even need to think." Perhaps a more effective instructional strategy would be to give the learner some information that will help them determine the correct value without giving them the correct value. The messages included the condition, the learner's action and the correct action. Perhaps stating that an error has occurred and give the child the first two parts of the statement would be effective. In addition, I found the textual information was difficult for them to understand. Perhaps a graphical approach that parallels the use of manipulatives for mathematical explanations would be most effective.

A couple of observations that I made during the field test are relevant to some unexpected aspects of POSIT. One observation was that the participants made a game out of anticipating where window buttons appeared before the window appeared. That is, POSIT saves the information about a problem as soon as the problem is completed. While the participants were waiting, they would move the mouse pointer to the location of the Yes button that would appear as soon as it was done saving. This was a popular game. The second game that was observed was played by only one child. POSIT does not check a value in an area until the user clicks another area (or the Done button when you are finished). P2 would type a value, align the mouse pointer in the next area, turn her back to the computer and push the mouse button. If it was a correct value, it would turn gray. If it was an incorrect value, there would be an arrow pointing at the area and a message on the screen. These games provided some unexpected motivating effects of POSIT.

The final comment has to do with the fact that 24% of the incorrect values entered by the participants either told them they did something they did not do or told them it did not know what they did. After one of these "bad" messages, P6 said, "You know, this program is pretty smart, but sometimes it is dumb."

Conclusions

Implementation

As the acronym POSIT implies, the purpose of this project was to design an interface that could be used to analyze children's subtraction solutions in process. The interface was successful in that it paralleled the algorithm that each child had learned. In addition, POSIT was "open" enough to allow children to perform mental calculations and not enter all possible correct states. The system's "open" interface also allowed children to enter values in any order with which they were comfortable. However, the interface does not allow for analysis of other algorithms. For example, some people have learned to make small "carry" marks next to the top number. There is no such area on the POSIT interface. Therefore, to the extent that POSIT allows any order and a subset of correct states, it is open. The fact that POSIT does not work well with other algorithms makes it a closed system.

A secondary purpose of this project was to develop a complete ITS on a microcomputer. Although the intent was to develop a system that would work with a standard MacIntosh configuration, this was not possible due to the size of the program. However, POSIT works well with a minimum of 2 megabytes of RAM and could work with a double floppy drive configuration. The two floppy drive system is, however, very slow. The program files would have to be read dynamically and this takes time. Therefore, the ideal machine would have to be configured with at least 2 megabytes of RAM and a hard disk drive. Given such a system, POSIT has been successfully implemented on a microcomputer.

The most successful aspect of this implementation is the model that was used to diagnose errors. The Errors of Omission and Commission were derived from the logical structure of subtraction and psychological constructs that would effect successful performance in subtraction. These error types accounted for over half of all errors made by the participants of the field test. Brown and Burton (1978) used an error pattern, borrow do not decrement, that would be the same as the Error of Omission, forgetting to decrement. However, the latter makes more sense in terms of what we know about the limits of short-term memory (Miller, 1956). The former would be placed into Brown and VanLehn's (1980) Repair Theory, but I am not sure how Repair Theory would explain this bug (perhaps under the category of slips). Therefore, the explanatory aspects of Errors of Omission and Commission are more appropriate in this case. This does not preclude the value of Repair Theory. Rather, Repair Theory was used as the basis for including the third category of errors, Errors of Creation.

Although the Errors of Creation did not account for many of the errors in the field test, a different population of users might have resulted in different frequencies. As was noted in the last section of the Results and Discussion section, the most common error on the pre-test was forgetting to decrement (an Error of Omission). Therefore, it is not surprising that this was the most common error during the field test and a group of participants who made many Errors of Creation on the pre-test could have drastically changed the frequency of errors on the field test. Further, the field test indicated that a variety of Errors of Creation might be included for the regrouping areas and make correct diagnosis more frequent. Therefore, the error categories used for the model of incorrect solutions more clearly organize errors in subtraction and have more meaningful instructional implications (forgetting to decrement can be explained in that way, rather than telling the child that he was using the borrow do not decrement error pattern).

One final comment should be made with regard to the model of incorrect solutions. The fact that POSIT has more information to examine than the error pattern systems

(Brown & Burton, 1978; Ohlsson & Langley, 1985; Young & O'Shea, 1981), allowed it to be more successful in determining incorrect solutions (76% on the current version, with potential of achieving 80 to 90% accuracy). The success of POSIT in determining the incorrect solution made by the child is the most important result of this project.

Finally, the Teach component of POSIT has implications for all software development. By using an on-line walkthrough to teach people how to use POSIT, a great deal of time was saved. This approach can be used for any software system. Granted, it takes time to develop and it takes up more file space, but it was a successful approach to training people to use the system.

Learning from POSIT

There were two approaches used in the field test to determine if children can learn from POSIT. One approach was to measure learning on POSIT and the other was to measure learning based on paper and pencil tasks. Both of these approaches indicated that children do learn from the system. The first approach indicated that there was a reduction in errors, a reduction in time taken on each step of the problem and a slight reduction in the use of the Help button. Taken as a whole, these results lend some support for the notion that children learn from POSIT. The second approach, although a weaker measure, also indicated that learning had occurred. However, how much of this learning can be directly attributed to POSIT is unclear.

Future Research

The obvious first move would be to make improvements to the model of incorrect solutions. These changes have been described and would make appropriate feedback by POSIT a more common occurrence.

The second direction to be taken as the result of this project is to design an experiment that can determine the effectiveness of the instruction provided by POSIT. Such an experiment might be designed on the basis of Bloom's (1984) 2-sigma problem (described elsewhere). That is, the immediate and appropriate feedback provided by POSIT could be compared to human tutors and these two strategies could be compared to the delayed feedback that is typical in classroom exercises (no feedback until after the exercises have been graded). A study such as this might be used to directly address the learning issue.

A third modification to POSIT might be made to the instructional presentation. Currently all feedback is described in a textual format. The content of this text includes the correct answer. A possible alternative approach is provide the explanatory text without the correct solution. The child would then have to determine the correct action based on the error statement. In addition, voice might be used to eliminate the problem of readability.

A second instructional strategy might be to use some form of graphical representation of the problem and its current solution state. This representation can then be used to describe the condition, the child's action and why the child's action could not work. This graphical representation could use animation to describe the specific action or actions taken by the child. This is a strategy commonly used in mathematics instruction, specifically the use of manipulatives. Of concern would be whether the graphical representation would be as effective as concrete materials.

The last instructional strategy to be discussed (although these are not the only ones which could be used) would be to branch off to set of problems that have been partially solved, but where the next step in the solution is the same as the step where the error has occurred. The child would then be asked to determine the correct value in each of these partial solutions until some correct criteria is met. At that point, the program would return control to the original problem.

Two approaches could be used to determine which strategy is best. One approach would make all three instructional strategies available and allow the learners to choose the instructional strategy. In this way, learner preferences for a particular instructional strategy could be determined. A second approach would be to make three versions of POSIT each with its own instructional strategy and determine which strategy is most effective.

As a final direction for future research, I think that the process-oriented analysis which is inherent in POSIT could be used in a variety of contexts. Certainly, most algorithmic learning in mathematics has areas for "scratch" work. These areas can be built into an interface and the process analyzed. In this way, the learning of algorithms can be made more efficient thereby allowing more class time for the teaching of more conceptually-oriented mathematics instruction. Regardless of the content of the instruction, if the content is a cognitive *process* it should be analyzed in *process*.

Footnotes

1. Fisher and Berliner (1985) edited a book which is a collection of articles focussing on instructional time. Especially interesting is Smyth (1985) who traces the history of this concept and discusses (among other things) the importance of pupil engagement.

2. Lisp is a computer programming language often used in the area of artificial intelligence.

3. It is quite difficult to describe a computer program that is not given a name. Further, Young and O'Shea never say what sort of production system they used to develop their system, but they denote their system as PS and I will follow suit.

4. Openness refers to the notion that the system allows a wide variety of solution strategies. A closed system, on the other hand, would have one model for the correct strategy and the student would be forced into this strategy.

5. Wenger (1987) describes the ideal model as one which "should include all the aspects of the student's behavior and knowledge that have repercussions for his performance and learning" (p. 16). A computer model will certainly be incomplete, but it should at least include aspects that are directly relevant to the child's learning and therefore, the ITS's teaching.

References

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*. *89*(4), 369-406.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review*. *94*(2), 192-210.

Anderson, J. R., Boyle, C. F. & Reiser, B. J. (1985). Intelligent tutoring systems. *Science*. *228*, 456-462.

Anderson, J. R. & Reiser, B. J. (1985). The lisp tutor. *Byte*, *10*(4), 159-175.

Anderson, J. R. & Skwarecki, E. (1986). The automated tutoring of introductory programming. *CACM*, *29*(9), 842-849.

Becker, H. J. (1987). Using computers for instruction. *Educational Computing*, *February*, 149-160.

Blando, J. A., Carlsen, W. C. & Sleeman, D. (1986, April). *Using an intelligent tutoring system to uncover errors of precedence*. Paper presented at the annual American Educational Research Association Conference, San Francisco.

Bloom, B. S. (1984). The 2-Sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, *13*(6), 4-16.

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

Brown, J. S. & Van Lehn K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, *4*, 379-427.

Burton, R. R. & Brown, J. S. (1982). An investigation of computer coaching for informal learning environments. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems*. London: Academic.

Diener, C. I. & Dweck, C. S. (1978). An analysis of learned helplessness: Continuous changes in performance, strategy, and achievement cognitions following failure. *Journal of Personality and Social Psychology*, *36*(5), 451-462.

Ericsson, K. A. & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data*. Cambridge, MA: The MIT Press.

Fisher, C. W. & Berliner, D. C. (1985). *Perspectives on instructional time*. NY: Longman.

Fitts, P. M. (1964). Perceptual-motor skill learning. In A. W. Melton (Ed.), *Categories of Human Learning*, NY: Academic Press.

Gagne, R. M. & Briggs, L. J. (1979). *Principles of instructional design.* (2nd. ed.) NY: Holt, Rinehart and Winston.

Goldman, S. R. & Pellegrino, J. W. (1987). Information processing and educational microcomputer technology: Where do we go from here? *Journal of Learning Disabilities, 20*(3), 144-154.

Hartley, J. R. & Sleeman, D. H. (1973). Towards intelligent teaching systems. *International Journal of Man-Machine Studies. 5*, 215-236.

Johnson, W. L. (1985). *Intention-based diagnosis of errors in novice programmers.* New Haven, CT: Department of Computer Science, Yale University (YALEU/CSD/RR #395).

Kheong, F. H. (1987). Computer-aided diagnosis and remediation in fractions. In J. C. Bergeron, N. Herscovics & C. Kieran (Eds.), *Proceedings of the Eleventh International Conference of the Psychology of Mathematics Education (Vol 1)*, (pp. 373-382), Montreal, Quebec, Canada: University of Montreal Press.

Kouba, V. L., Brown, C. A., Carpenter, T. P., Lindquist, M. M., Silver, E. A. & Swafford, J. O. (1988). Results of the fourth NAEP assessment of mathematics: Number, operations, and word problems. *Arithmetic Teacher, 35*(8), 14-19.

Lawler, R. W. & Yazdani, M. (1987). *Artificial intelligence and education (Vol. 1).* Norwood, NJ: Ablex.

Mayer, R. E. (1983). *Thinking, problem solving, cognition.* NY: Freeman and Company.

Miller, G. A., (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review, 63*, 81-97.

National Council of Teachers of Mathematics (1989). *Curriculum and instruction standards for school mathematics.* Reston, VA: NCTM.

Newell, A. & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Ohlsson S. & Langley P. (1985). *Identifying solution paths in cognitive diagnosis.* Pittsburgh, PA: The Robotics Institute, Carnegie Mellon University (CMU-RI-TR-85-2).

Orey, M. A. (1987). *What do error patterns tell us?* Unpublished master's thesis, VPI&SU, Blacksburg, VA.

Orey, M. A. & Miller, D. P. (1988). Diagnostic computer systems for arithmetic. *Computers in the Schools. 4*(3/4), 139-149.

Payne, J.N., Beardsley, L.M., Bunch, B.H., Carter, B.B., Coburn, T.G., Edmonds, G.F., Payne, R.C., Rathmell, E.C. & Trafton, P.R. (1985). *Mathematics.* (2nd. ed.). NY: Harper and Row.

Reiser, B. J. , Anderson, J. R. & Farrell, R. G. (1985). Dynamic student modeling in an intelligent tutor of LISP programming. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* 8-14.

Rosenberg, R. (1987). A critical analysis of research on intelligent tutoring systems. *Educational Technologist. 27*(11), 7-13.

Rummelhart, D. E. & Norman, D. A. (1978). Accretion, tuning, and restructuring: Three modes of learning. In J. W. Cotton & R. Klatzsky (Eds.), *Semantic factors in cognition.* Hillsdale, NJ: Erlbaum.

Salomon, G. & Gardner, H. (1986). The computer as educator: Lessons from television research. *Educational Researcher, 15*(1), 13-19.

Smyth, W. J. (1985). A context for the study of time and instruction. In C. W. Fisher & D. C. Berliner (Eds.), *Perspectives on instructional time.* NY: Longman.

Thompson, C. S. & Rathmell, E. C. (1988). NCTM's standards for school mathematics, K-12. *Mathematics Teacher, 81*(5), 348-351.

Torgesen, J. K. & Young, K. A. (1983). Priorities for the use of microcomputers with learning disabled children. *Journal of Learning Disabilities, 16*(4), 234-237.

Underhill, B., Uprichard, E. & Heddens, J. (1980). *Diagnosing mathematical difficulties.* Columbus, OH: Merrill Publishing.

Van Lehn K. (1982). Bugs are not enough: Empirical studies of bugs, impasses, and repairs in procedural skills. *Journal of Mathematical Behavior, 3,* 3-72.

Weiss, I. R. (1987). *Report of the 1985-86 national survey of science and mathematics education.* North Carolina: Research Triangle Institute (RTI/2938/00-FR).

Wenger, E. (1987). *Artificial intelligence and tutoring systems.* Los Altos, CA: Morgan Kaufmann.

Will, M. (1986). *Educating students with learning problems: A shared responsibility.* Washington, D.C.: Office of special education and rehabilitative services.

Young, R. M. & O'Shea, T. (1981). Errors in children's subtraction. *Cognitive Science, 5,* 153-177.

Appendix A. Random problem generators in ascending level of difficulty.

<u>2-Digit</u>
no regroup
regroup

<u>3-digit</u>
no regroup
regroup ones
regroup tens
regroup both
zero in tens

<u>4-digit</u>
no regroup
regroup ones
regroup tens
regroup hundreds
regroup ones and tens
regroup tens and hundreds
regroup ones and hundreds
regroup ones, tens and hundreds
zero in tens
zero in hundreds
zero in both

Appendix B. Complete set of possible states and their corresponding implications.

Area	Possible States
1	Z+10
2	Y-1, Y+10, (Y-1)+10, 9,10
3	X-1, X+10, (X-1)+10, 9,10,NON-EXIST
4	W-1,NON-EXIST
5	Z-D, (Z+10)-D
6	Y-C, (Y-1)-C. (Y+10)-C, ((Y-1)+10)-C, 9-C, 10-C, NOTHING
7	X-B, (X-1)-B, (X+10)-B, ((X-1)+10)-B, 9-B, 10-B, NOTHING, NON-EXIST
8	W-A, (W-1)-A, NOTHING, NON-EXIST
9	((Y-1)+10), 9, NON-EXIST
10	((X-1)+10), 9, NON-EXIST

NOTE: Non-exist is for conditions where there are fewer than 4-digits in the problem. Nothing is for the condition when there are leading zeros.

Given the above states, these are the implications that are present when a particular state is reached.

Definitions:

Pass - Pass unresolved implications to area above, make current area inactive
Resolve - Resolve outstanding implications and make area inactive
Increment - Add 10
Decrement - Subtract 1

Area	State Change	Implication
1	Z+10	2 or 9-Decrement, 1-Resolve
2	Y-1	1-Increment, 9-Pass, 2-Resolve
	(Y-1)+10	1-Increment, 3 or 10-Decrement, 2,9-Resolve
	Y+10	3 or 10-Decrement, 2-Pass
	9	1-Increment, 3 or 10-Decrement, 2,9-Resolve
	10	3 or 10-Decrement, 2-Pass
3	X-1	2 or 9-Increment, 3-Pass
	(X-1)+10	2 or 9-Increment, 4-Decrement, 3,10-Resolve
	X+10	4-Decrement, 3-Pass
	9	2 or 9-Increment, 4-Decrement, 3,10-Resolve
	10	4-Decrement, 3-Pass
	NON-EXIST	3,4,7,8,9,10-Resolve
4	W-1	3 or 10-Increment, 4-Resolve
	NON-EXIST	4,8,10-Resolve
5	Z-D	2 or 9-Not Decrement, 1,5-Resolve
	(Z-1)-D	2 or 9-Decrement, 1,5-Resolve

6	Y-C	1-Not Increment, 3 or 10 Not Decrement, 2,6,9-Resolve
	(Y-1)-C	1-Increment, 3 or 10-Not Decrement, 2,6,9-Resolve
	((Y-1)+10)-C	1-Increment, 3 or 10-Decrement, 2,6,9-Resolve
	(Y+10)-C	1-Not Decrement, 3 or 10-Decrement, 2,6,9-Resolve
	9-C	1-Increment, 3 or 10-Decrement, 2,6,9-Resolve
	10-C	1-Not Decrement, 3 or 10-Decrement, 2,6,9-Resolve
	NOTHING	2,3,4,6,7,8,9,10-Resolve
7	X-B	2 or 9-Not Increment, 3,4,7,10-Resolve
	(X-1)-B	2 or 9-Increment, 3,4,7,10-Resolve
	((X-1)+10)-B	2 or 9-Increment, 4-Decrement, 3,7,10-Resolve
	(X+10)-B	2 or 9-Not Decrement, 4-Decrement, 3,7,10-Resolve
	9-B	2 or 9-Increment, 4-Decrement, 3,7,10-Resolve
	10-B	2 or 9-Not Decrement, 4-Decrement, 3,7,10-Resolve
	NOTHING	3,4,7,8,10-Resolve
	NON-EXIST	3,4,7,8,10-Resolve
8	W-A	3 or 10-Not Increment, 4,8-Resolve
	(W-1)-A	3 or 10-Increment, 4,8-Resolve
	NOTHING	4,8-Resolve
	NON-EXIST	4,8-Resolve
9	(Y-1)+10	1-Increment, 3 or 10-Decrement, 2-Resolve
	9	1-Increment, 3 or 10-Decrement, 2-Resolve
	NON-EXIST	3,4,7,8,9,10-Resolve
10	(X-1)+10	2 or 9-Increment, 4-Decrement, 3-Resolve
	9	2 or 9-Increment, 4-Decrement, 3-Resolve
	NON-EXIST	4,7,8

Appendix C. The pre and post-tests.

NAME: Pre-test

$\begin{array}{r} 647 \\ -\ \underline{\ \ 45} \end{array}$	$\begin{array}{r} 885 \\ -\ \underline{205} \end{array}$	$\begin{array}{r} 83 \\ -\ \underline{44} \end{array}$
$\begin{array}{r} 8305 \\ -\ \underline{\ \ \ \ 3} \end{array}$	$\begin{array}{r} 50 \\ -\ \underline{23} \end{array}$	$\begin{array}{r} 562 \\ -\ \underline{\ \ \ 3} \end{array}$
$\begin{array}{r} 742 \\ -\ \underline{136} \end{array}$	$\begin{array}{r} 106 \\ -\ \underline{\ \ 70} \end{array}$	$\begin{array}{r} 716 \\ -\ \underline{598} \end{array}$
$\begin{array}{r} 1564 \\ -\ \underline{\ \ 887} \end{array}$	$\begin{array}{r} 6591 \\ -\ \underline{2697} \end{array}$	$\begin{array}{r} 311 \\ -\ \underline{214} \end{array}$

$$\begin{array}{r} 1813 \\ -\ \underline{\ \ 215} \end{array} \qquad \begin{array}{r} 102 \\ -\ \underline{\ \ 39} \end{array} \qquad \begin{array}{r} 9007 \\ -\ \underline{6880} \end{array}$$

$$\begin{array}{r} 4015 \\ -\ \underline{\ \ 607} \end{array} \qquad \begin{array}{r} 702 \\ -\ \underline{108} \end{array} \qquad \begin{array}{r} 2006 \\ -\ \underline{\ \ \ \ 42} \end{array}$$

$$\begin{array}{r} 10012 \\ -\ \underline{\ \ \ \ 214} \end{array} \qquad \begin{array}{r} 8001 \\ -\ \underline{\ \ \ \ 43} \end{array}$$

NAME: Post-test

$\begin{array}{r} 536 \\ -\ \ 34 \\ \hline \end{array}$	$\begin{array}{r} 773 \\ -\ 303 \\ \hline \end{array}$	$\begin{array}{r} 64 \\ -\ 45 \\ \hline \end{array}$
$\begin{array}{r} 7206 \\ -\ \ \ \ 4 \\ \hline \end{array}$	$\begin{array}{r} 70 \\ -\ 43 \\ \hline \end{array}$	$\begin{array}{r} 453 \\ -\ \ \ 4 \\ \hline \end{array}$
$\begin{array}{r} 635 \\ -\ 226 \\ \hline \end{array}$	$\begin{array}{r} 208 \\ -\ \ 60 \\ \hline \end{array}$	$\begin{array}{r} 817 \\ -\ 498 \\ \hline \end{array}$
$\begin{array}{r} 1784 \\ -\ \ 896 \\ \hline \end{array}$	$\begin{array}{r} 6782 \\ -\ 3887 \\ \hline \end{array}$	$\begin{array}{r} 456 \\ -\ 257 \\ \hline \end{array}$

$\begin{array}{r} 3416 \\ -\ \underline{\ \ 119} \end{array}$	$\begin{array}{r} 205 \\ -\ \underline{\ \ 49} \end{array}$	$\begin{array}{r} 8006 \\ -\ \underline{5770} \end{array}$
$\begin{array}{r} 3014 \\ -\ \underline{\ \ 406} \end{array}$	$\begin{array}{r} 804 \\ -\ \underline{307} \end{array}$	$\begin{array}{r} 1005 \\ -\ \underline{\ \ \ \ 63} \end{array}$
$\begin{array}{r} 10024 \\ -\ \underline{\ \ \ \ 327} \end{array}$	$\begin{array}{r} 7002 \\ -\ \underline{\ \ \ \ 54} \end{array}$	