# The Role of Automation in Education: Now and in the Future

**Joseph M. Scandura**

**Director of Research, MERGE Research Institute and**

**Emeritus Professor, University of Pennsylvania**

According to Wikipedia "Automation is a step beyond mechanism. Whereas mechanization provided human operators with machinery to assist them with the muscular requirements of work, automation greatly reduces the need for human sensory and mental requirements as well. In this context, Artificial Intelligent (AI) was founded on the claim that a central property of human intelligence can be so precisely described that it can be simulated by a machine. Proponents have long claimed that increases in computational power will eventually overtake the human mind. IBM's Big Blue beating Chess masters is often sighted to support this claim. On the other hand, most AI research has become increasingly technical and specialized.

Progress is being made in subfields: solutions to specific problems can be automated. This is a pattern that has been replicated in almost every software intensive application area. Who today would compute taxes using paper and pencil? Keep records on a rolodex? … We have immediate access to almost any information in databases, instant communication throughout the world and the ability to quickly find information on almost any topic – at least if it occurred or was documented after the advent of the world wide web.

Intelligent behavior has not happened, however, except in a very special sense: More and more things that humans used to have to do themselves can be automated on computer. increasingly complex tasks have been automated -- not to the extent that they can be done as well as humans, but better.

My goal today is to draw and develop parallels to education. Major attention is being given to immersive, often game-like environments. Students are placed in situations -- and allowed to explore on their own or with various kinds of hints (typically called "scaffolding"). The big questions here are what kinds of scaffolding will be of (most) help and when should it be given?

Other tools such as Texas Instrument's TI-Nspire tackle the problem from the other end. Rather than hints, calculators serve as tools that facilitate problem solving. They serve as prerequisites -- as foundational skills on which learners may build.

Scaffolding and prerequisites both play a central role in all learning systems. The main problem is that good tutoring systems have been difficult and expensive to build. Moreover, their educational benefits are difficult and expensive to evaluate. Determining effectiveness and efficiency invariably requires direct (and often expensive) empirical evaluation. The results are rarely if ever as good as what a human tutor can do.

Good instructional design models help. Among other things they help identify what must be mastered for success and what can be assumed on entry. Computer Based Instruction (CBI) systems build on assumed prerequisites and are directed at what must be learned. After years of effort, beginning with Control Data's work (under the leadership of William Norris) in the early 1960s, the best CBI is limited to providing pretests to identify areas of weakness, providing instruction aimed at deficiencies and following up with post tests to determine how much has been learned.

ALEKS is one of the better commercially available CBI systems. In ALEKS and other advanced CBI systems (e.g., Paquette, 2007, 2009) to-be-acquired knowledge is represented in terms of relational models.

ITS research goes further, attempting to duplicate or model what a good tutor can do – by adjusting diagnosis and remediation dynamically during instruction. ITS focus on modeling and diagnosing what is going on in learner minds (e.g., Anderson, 1993; cf. Koedinger et al, 1997; Scandura et al, 2007). Assumptions are made both about what knowledge (e.g., productions) might be in learner minds and learning mechanisms controlling the way those productions are used in producing behavior and learning.

Identifying the productions involved in any given domain is a difficult task. Specifying learning mechanisms is even harder. Recognizing these complexities Carnegie Learning credits Anderson's evolving ACT theories, but increasingly has focused on integrating ITS with print materials to make them educationally palatable.

The difficulties do not stop there. Ohlsson noted as early as 1987 that specifying remedial actions – what to teach is much harder than modeling and diagnosis. As in CBI, pedagogical decisions in ITS necessarily depend on the subject matter being taught – on semantics of the content. Each content domain requires its own unique set of pedagogical decisions. From their inceptions, the Holy Grail in both CBI and ITS is to duplicate what good teachers do. As shown by Bloom (1984) the best human tutors can improve mastery in comparison to normal instruction by 2 sigmas. This goal has been broadly influential but never achieved through automation. The limited success of CBI, combined with the complexities and cost inefficiencies of ITS have reduced effort and research support for both CBI and ITS.

I will show that these trends are premature. Advances in SLT and AuthorIT and TutorIT technologies based thereon **make it possible not only to duplicate human tutors in many areas but to do better.** Today, for example, few doubt we can build tutoring systems that teach facts as well or better than humans. "Flash cards", for example, could easily be replaced by computers -- with more efficiency and certain results.

Today, I will go further:

a) I will show that AuthorIT makes it possible to create and that TutorIT now makes it possible to deliver highly adaptive (and configurable) tutoring systems that can do as good or better job on well-defined math skills.

b) I will show why and in what sense TutorIT tutorials can guarantee mastery of such skills.

c) I will show why TutorIT tutorials can be developed cost effectively – at half the cost of traditional CBI development.

d) I will show how TutorIT tutorials can gradually be extended to support the development and delivery of higher as well as lower order knowledge.

e) I will demonstrate why TutorIT tutorials can be expected to produce as good or better learning than most humans.

My paper is organized as follows:

1. I provide some background and summarize recent advances in knowledge representation and Structural Learning Theory (SLT) offering a theoretically rigorous, empirically sound foundation for building highly adaptive tutoring systems.

2. I first show why these advances make it possible to exceed human tutoring on well defined knowledge.
3. I then show how AuthorIT makes it possible to develop highly effective TutorIT tutorials at greatly reduced cost – even less than commercial development. I will also demonstrate how TutorIT math tutorials work and explain why they can do as good if not better job than most human tutors, how they can be configured at no additional cost to meet alternative needs – e.g., to serve as diagnostic systems, and how they can be used to reliably compare alternative pedagogies.
4. Finally, I talk about the future: What needs to be done to support ill-defined domains where higher order knowledge plays a central role? I show how AuthorIT and TutorIT can be extended to support adaptive tutoring on higher order learning – and why such tutoring systems may be expected to do as well, to even supersede human tutors.

**Background**

In the 1960s, there was a disconnect in educational research and research in subject matter (math) education. Educational research focused on behavioral variables: exposition vs. discovery, example vs. didactic, demonstration vs. discussion, text vs. pictures, aptitude-treatment interactions, etc. (cf. Scandura, 1963, 1964a,b). Subject matter variables were either ignored or limited to such things as simple, moderate, difficult. Little attention was given to what makes content simple, moderate or difficult. Conversely, research in subject matter (e.g., math) education, focused primarily on content (reading, writing, arithmetic skills, algebraic equations, proof, etc.).

In same time period, instructional design focused on what was to be learned and prerequisites for same. Task analysis focused initially on behavior – on what learners need to do (Miller, 1959; Gagne, 1966). In my own work, this focus morphed into cognitive task analysis – on what learners must learn for success (e.g., Scandura, 1970, 1971, Durnin & Scandura, 1973). My parallel work in experimental psychology (Greeno & Scandura, 1966; Scandura & Roughead, 1967) in the mid 1960s added the critical dimension of behavior to the equation.

Structural Learning grew out of this disconnect, with the goal of integrating content structure with human cognition and behavior. Structural Learning Theory (SLT) was first introduced as a unified theory in 1970 (published in Scandura, 1971a). SLT's focus from day one (and the decade of research on problem solving and rule learning which preceded it) was on what must be learned for success in complex domains, ranging from early studies of problem solving and rule learning (Roughead & Scandura, 1968; Scandura, 1963, 1964a,b, 1973, 1977) to Piagetian conservation (Scandura & Scandura, 1980), constructions with straight edge and compass, mathematical proofs and critical reading (e.g., Scandura, 1977).

This research was focused on the following four basic questions [with their evolution from 1970 → Now]:

- Content: What does it mean to know something? And how can one represent knowledge in a way that has behavioral relevance?

  [1970: Directed graphs (flowcharts) → **Now: Abstract Syntax Trees (ASTs) & Structural Analysis (SA)]**

- Cognition: How do learners use and acquire knowledge? Why is it that some people can solve problems whereas others cannot?

  [1970: Goal switching → **Now: Universal Control Mechanism (UCM)]**

- Assessing Behavior: How can one determine what an individual does and does not know?

  [1970: Which paths are known → **Now: which nodes in AST are known (+), -, ?]**

- Instruction: How does knowledge change over time as a result of interacting with an external environment?

  [1970: Single level diagnosis & remediation → **Now: Multi-level inferences about what is known and what needs to be learned**]

Higher order and lower order knowledge played a central role in this research from its inceptions – with emphasis on the central role of higher order knowledge in problem solving (Scandura,

1971, 1973, 1977). Early SLT research also focused heavily on indentifying what individual learners do and do not know relative to what needed to be learned (e.g., Durnin & Scandura, 1974; Scandura, 1971, 1973, 1977).

Deterministic theorizing was a major distinguishing feature of this research (Scandura, 1971). I was focused, even obsessed with understanding, predicting and (in so far as education is concerned) controlling how individuals solve problems. Despite considerable training in statistics and having conducted a good deal of traditional experimental research (e.g., Greeno & Scandura, 1966; Scandura & Roughead, 1967; Scandura, 1967), I found unsatisfying more akin to what had been accomplished in physics centuries earlier (cf. Scandura, 1974a).

SLT was unique when introduced, and raised considerable interest both in the US and internationally (Scandura, 1971a, 1973, 1977). Literally hundreds of CBI programs based on SLT were developed later in the 1970s and early 1980s, and many sold for decades.

Nonetheless, ITS largely ignored this research and focused on later work in cognitive psychology (Anderson et al, 1990, 1993) and especially the Carnegie school of artificial intelligence based on production systems (esp. Newell & Simon, 1972).

By the mid-1970s, cognitive psychology also discovered the importance of content, often equating theory with alternative ways of representing knowledge. Research focused largely on what (productions or relationships) might be in learner minds and comparing fit with observable behavior. Experimental studies followed the traditional statistical paradigm.

Similarly, most CBI development was heavily influenced by Gagne's work in instructional design (1965), along with that of Merrill and his students, 1994). The restricted focus of Reigeluth's (1983, 1987) influential books on Instructional Design largely eliminated or obscured some of SLT's most Important features, most notably its focus on precise diagnosis and higher order learning and problem solving. With essential differences requiring significant study, the long and short of it is that other than our own early tutorials (which made small publisher Queue one of Inc Magazine's 100 fastest growing small businesses), SLT failed to significantly inform on-going research in either CBI or ITS. After the interdisciplinary doctoral program in structural learning I developed at Penn was eliminated in the early-mid 1970s, SLT became a little understood historical curiosity.

With recent publications in TICL, depth of understanding in ITS, CBI and SLT has increased in recent years (Mitrovic & Ohlsson, 2007; Paquette, 2007; Scandura, 2007), including their respective advantages and limitations (Scandura, Koedinger, Mitrovic & Ohlsson, Paquette, 2009). Advances in the way knowledge is represented in SLT has the potential of revolutionizing the way tutoring systems are developed, both now and in the future. SLT rules were originally represented as directed graphs (e.g., Scandura, 1971a, 1973). Directed graphs (equivalent to Flowcharts) make it possible to assess individual knowledge. They have the disadvantage, however, of forcing one to make a priori judgments about level of analysis. They also make it difficult to identify subsets of problems associated with various paths in those graphs.

Having spent two decades in software engineering (e.g., Scandura, 1991, 1994 1995, 1999, 2001), it became increasingly apparent that a specific form of Abstract Syntax trees (ASTs) offered a long sought solution. ASTs are a precise formalism derived from compiler theory and that are widely used in software engineering. To date, ASTs have had almost no impact on knowledge representation, ITS or CBI. However, we will see that they do indeed have very significant advantages in SLT.

I have recently documented the current form of SLT in some detail (Scandura, 2007). Readers are encouraged to review the material therein on Knowledge Representation along with the published dialog on the subject which followed (Scandura, Koedinger, Mitrovic & Ohlsson and Paquette, 2009).

I focus in the next section on what is most unique about knowledge representation in SLT along with why and how it offers major advantages in developing adaptive tutoring systems.

## 1. Theoretical Advances: Well-Defined Knowledge

There have been three fundamental advances in SLT in recent years. First is in the way knowledge is represented. SLT rules were originally represented as directed graphs (Flowcharts). They are now represented in terms of Abstract Syntax Trees (ASTs). Second is formalization of a key step in Structural (domain) Analysis (SA), enabling the systematic identification of higher order SLT rules that must be learned for success in ill-defined domains. Third is the complete separation of SLT's control mechanism from higher order knowledge. These advances distinguish knowledge representation in SLT from all others, and have fundamental implications for building adaptive tutoring systems.

In this section we consider the first advance: SLT rules have long been used to represent to-be-acquired knowledge in well-defined domains. While retaining the advantages of directed graphs, representing SLT rules in terms of Abstract Syntax Trees (ASTs) offers a number of critically important benefits.

Not only do they offer a way to assess individual knowledge (as did directed graphs), but AST-based SLT rules also provide a perfectly general way to automatically both generate test problems and the solutions to those test problems. As we shall see, they also make it possible to simultaneously represent knowledge at any number of levels of analysis.

*Precision*.-- The major reason adaptive tutoring systems have been so difficult and expensive to develop is that pedagogical decision making has been so time consuming and expensive. This is equally true whether tutoring systems are based on traditional CBI (cf. Paquette, 2007) or ITS (cf. Mitrovic & Ohlsson, 2007).

In CBI the focus is on what must be learned. Better CBI systems invariably are based on some combination of hierarchical and/or relational analysis. Hierarchical representations have an important advantage: Hierarchies inherently arrange content in the order in which content must be learned. Content higher in a hierarchy necessarily incorporates lower order content, a fact that has direct and important implications for both testing and teaching.

The problem is twofold:

> (1) not everything can be represented hierarchically using current decomposition methods (Scandura, 2007) and
>
> (2) informal hierarchical representation is not sufficiently precise to automate decision making without direct attention to the meaning of the content.

I don't want to don't have time to repeat here what has already been published. On the other hand, I must call special attention to one key idea, an idea that makes it possible to develop adaptive tutoring systems that can both: a) be developed at lower cost and b) guarantee learning.

Specifically, Structural (domain) Analysis (e.g., Scandura, 2007) makes it possible not only to represent all behavior hierarchically, but to do so precisely that inherent relationships are exposed.

It is well know known that many ideas can be refined into components or categories. Components and categories are fundamental: Component refinements involve breaking sets into to their elements. Category refinements involve breaking sets into subsets. For example, the set of animals can be refined into elements -- individual animals in the set. The set of animals also can be refined into categories: dogs, cats, whales, etc.

Consider column subtraction: We begin with a subtraction problem. Subtraction problems typically are refined first into elements, the columns that make up a subtraction problem. (Because the number of columns in a subtraction problem may vary, I have called this variation

a “prototype” refinement, wherein each prototype, or column, has the same structure.) Columns, in turn, may be refined into categories, columns where the top number is greater than or equal to the bottom number and columns where the top number is less than the bottom number.

The same idea applies generally: Consider a “house”. Houses consist of sets of rooms, room elements. Rooms in turn can be categorized by their size, or their use, or by any number of other distinctions.

As detailed below, component and category refinements have direct counterparts in corresponding solution procedures. Again, consider column subtraction. Here, the initial procedural refinement is a Repeat-Until loop. Loops in procedures correspond precisely to Prototype refinements in data: Compute the answer to each column in turn until there are no more columns. The next procedural refinement is an IF..THEN selection. Selection refinements in procedures correspond to Category refinements in data. In subtraction, different processes are required when the top number is greater than or equal to the bottom number and when this is not the case.

Unfortunately, component and category refinements are not sufficient. Other kinds of “refinements” involve (more general) relationships – for example, whether the top digit is greater than or equal to the bottom digit. Mating involves a relationship between two animals -- male and female.

Simple relationships are fine when they are immediately understandable and unambiguous. In many cases, however, they are not. None of us, for example, would any problem writing the numeral “5”. Ask most four or five year olds, however, and the story is likely to be very different. Writing the numeral “5” requires a precise set of constructions involving straight and curved line segments.

Relational models can easily represent the relationships between such line segments. Indeed, everything can be represented in terms of relationships. The problem is twofold. The number of relationships increases rapidly as domains become increasingly complex. In complex domains, relationships on relationships can extend geometrically without bound.

Relational representations suffer from an additional problem (beyond the sheer number of relationships). Whereas component and category refinements may be repeated indefinitely, this is not possible with (non-unary) relationships. Every relationship (relational refinement) must be considered anew. There is no systematic way to represent given (non-unary) relationships in terms of simpler elements.

Knowledge representation using ASTs solves this problem. There is a fundamental mathematical equivalence between relations and functions. Each and every relationship can be represented by at least one function, or procedure, having its own inputs and outputs. For example, relationships between straight and curved line segments comprising the numeral “5” can be viewed as a procedure operating on such segments. These procedures in turn can be refined as the originals.

Why is this important? Consider the following. If we subject Column Subtraction to Structural Analysis, we are going to end up with terminal elements requiring such things as the child’s ability to write the numeral “5” (and “0”, “1”, “2”, …). No matter what is being learned there will always be things that learners must know on entry. Young children, for example, learn early on to do such things as write the numeral “5”. What is being learned here is not a relationship. Rather, it is an SLT rule that takes line segments as input and generates the numeral “5”.

Prerequisite SLT rules, in turn, can be refined as any other. The refinement process can be repeated indefinitely. No matter how complex the subject matter, or how naïve the target population, it is always possible to represent the knowledge necessary for success in hierarchical form. The introduction of what I have called "dynamic" refinements, along with component and category refinements, closes the loop. It is now possible to represent what needs to be learned in any domain in whatever detail may be necessary (and desirable).

*NOTE: It is worth noting incidentally that representing relationships as functions is equivalent in software engineering to introducing the notion of a "callback". Just as one may introduce functions operating on parameters in a dialog box, one can introduce functions generating outputs from inputs in a relationship.*

For those not mathematically inclined, all this may seem like a technical truism with little practical significance. In fact, however, this technical truism has fundamental practical significance. AST hierarchies provide a perfectly general way to define pedagogical decisions. All pedagogical decisions in SLT can be based entirely on the structure of to-be-learned SLT rules. This can all be done independently of content semantics.

Indefinite refinement makes it possible to define what needs to be learned with whatever precision may be necessary to make contact with knowledge available to any population of learners, no matter how naïve they might be initially.

Full hierarchical representation makes it possible to quickly determine the status of any individual's knowledge at each point in time (relative to a SLT rule hierarchy), and to provide the instruction necessary to advance. Given full analysis, empirical research (e.g., Scandura, 1970, 1971, 1973, 1974a, 1977' Durnin & Scandura, 1973) demonstrates that testing on a single test item is sufficient to determine whether a learner has mastered any given sub tree in an SLT rule.

It is not always feasible, however, nor necessary to undertake complete analysis. Nonetheless, even incomplete hierarchical analysis is better than none. Incomplete hierarchies provide a beginning -- a starting point that can be improved incrementally as time, resources and the importance of any particular tutoring system demands.

It is always possible to build effective tutoring systems by introducing a safety factor (Scandura, 2005) – as engineers do in designing a bridge. Instead of requiring a single success corresponding to any terminal node (in an SLT rule hierarchy) one can require any number of successes. This makes it possible in principle to guarantee learning.

*Efficiency of Development*.-- A major limitation of adaptive tutoring systems is that they have been hard to build. Defining (and implementing) pedagogical decisions -- what to test or teach and when -- is one of the most expensive, time consuming and error prone tasks required (Mitrovic & Ohlsson, 2007; Koedinger & Ohlsson, 2009). This perhaps is the primary reason so few truly adaptive tutoring systems exist after over 50 years of generous federal support.

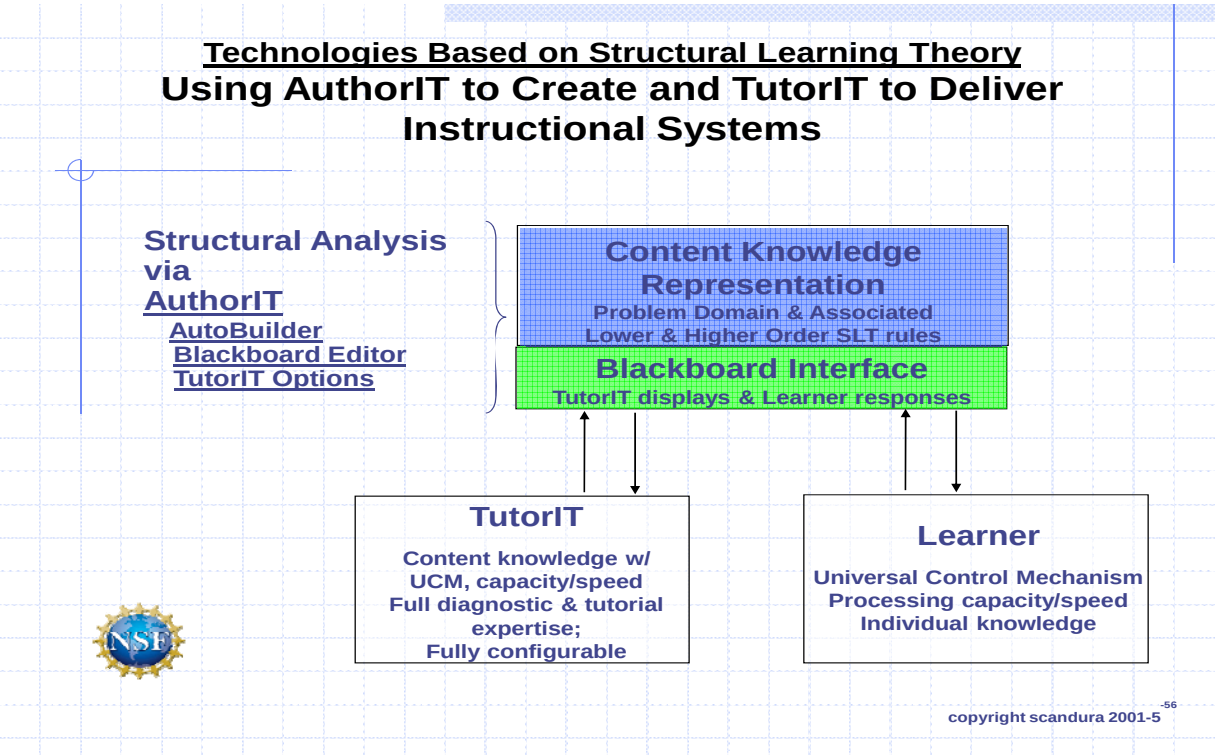
By way of contrast, I will show how TutorIT makes all pedagogical decisions automatically – based entirely on the hierarchical structure of SLT rules representing what is to be learned.

Hierarchical representation has a further not inconsequential benefit. It is easy to define any number of pedagogical theories as to how best to promote learning. Specifically, I will show how TutorIT can easily be configured to deliver instruction in accordance with a variety of pedagogical philosophies. In all cases, TutorIT effectively eliminates the need to program pedagogical decisions (Scandura 2005, 2007, 2009). Cost savings have been estimated at between 40 and 60% (cf. Foshay & Preese, 2005, 2006; Scandura, 2006a,b).

In short, guaranteed results at lower cost – a combination that should be hard to resist.

**Current Status of TutorIT: Guaranteed Learning and Lower Cost**

Our AuthorIT authoring and TutorIT delivery systems currently support the development and delivery of well defined knowledge (Scandura, 2005). The long term goal, however, is to realize the full potential of SLT.




A. Given a well defined problem domain, AuthorIT includes the following:

1. AutoBuilder, a tool for systematically representing knowledge as an SLT rule, including both the procedural Abstract Syntax Tree (AST) and the AST data structure on which it operates. Procedural ASTs in AutoBuilder are visually represented as Flexforms (below). Each node in a Flexform represents a specific part of the to-be-acquired knowledge.
2. Blackboard Editor, a tool for creating and laying out schemas representing problems in the domain. Blackboard serves as the interface through which learners and TutorIT interact.
3. AutoBuilder also is used to assign instruction, questions, positive feedback and corrective feedback to individual nodes in the Flexform. This information may include text, graphics, sound and/or other supporting media.
4. Options, a dialog (tool) used to define how TutorIT is to interact with learners. Options include variations on delivery modes ranging from highly adaptive to diagnostic, to simulation to practice.

B. TutorIT takes the above produced with AuthorIT and interacts with learners as prescribed in the Options Tool. I will show how TutorIT's adaptive mode works below. But, first let's review the development process.

*Representing Well defined Knowledge*.-- TutorIT development begins by representing to be learned knowledge as an SLT rule. As required by SLT (Scandura, 2005), AutoBuilder makes it possible to represent knowledge with arbitrary degrees of precision.

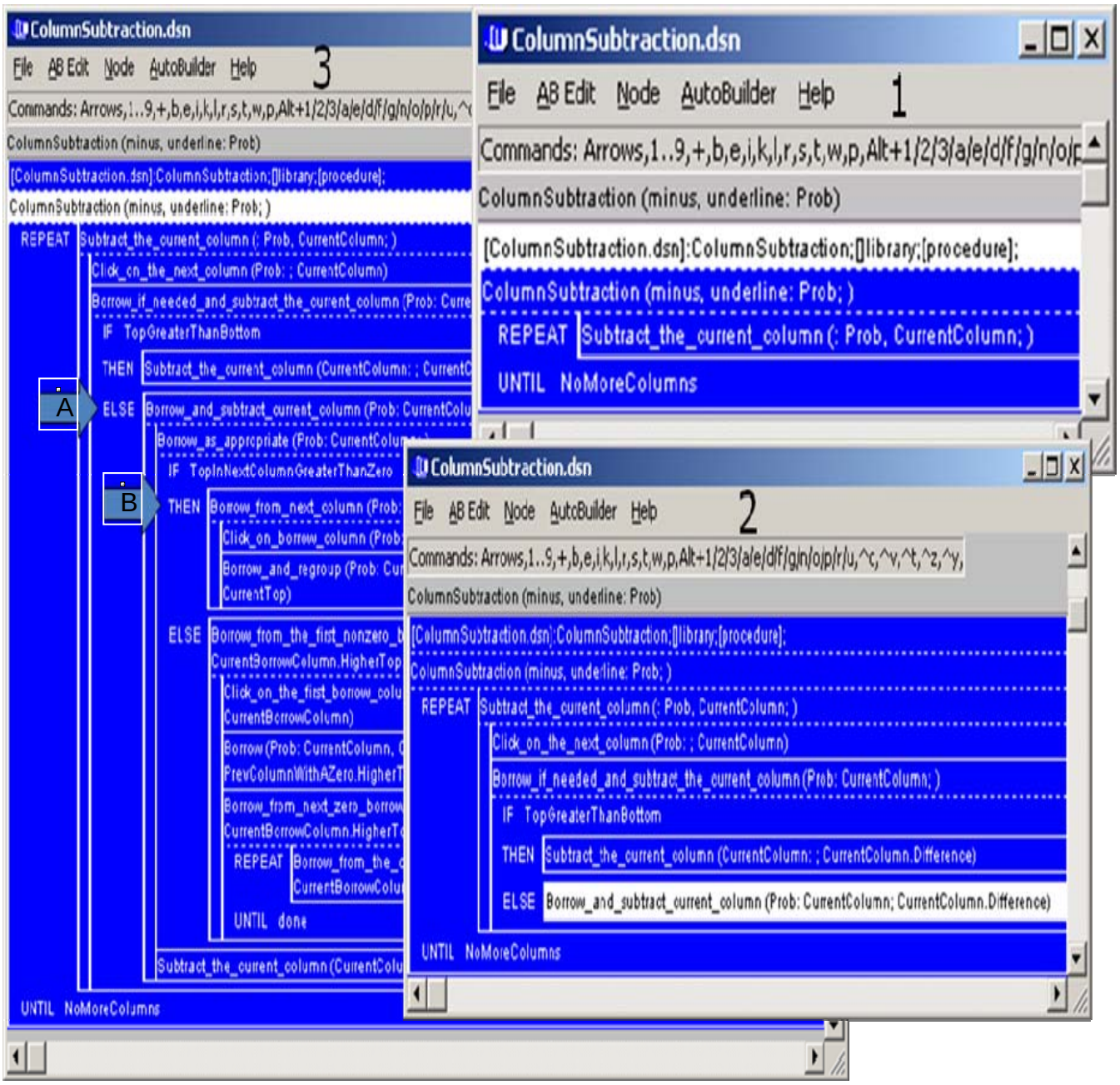



*Fig. 1. Successive levels of procedural refinement in Column Subtraction. This Flexform shows all levels refinement, from the highest levels of abstraction to the point where terminal nodes correspond to presumed prerequisites. In column subtraction these prerequisites include basic subtraction facts, ability to compare numbers as to size, etc. Students are tested on entry to ensure that they have mastered these prerequisites*.

Each node in the Flexform represents to-be-acquired knowledge at a specific level of abstraction. For example,

A. "*Borrow_and_subtract_the_current_column*" (to the right of the first "*ELSE*" in Fig. 1 represents the knowledge necessary for computing the difference in any column when the top digit is less than the bottom digit.

B. Subordinate nodes like "*Borrow_from_next_column*" provide increasingly more specific information.

Parameters of these operations representing data on which these operations act also are arranged hierarchically. Operation A, for example, operates on *"Prob"* and *"CurrentColumn". "Prob"* represents an entire subtraction problem. *"CurrentColumn"* represents columns in such problems. Operation B also includes *"ReducedTop", "Slashtop", "CurrentBorrowColumn"* and *"BorrowedDigit".*

In this context, AuthorIT's AutoBuilder component imposes consistency requirements on successive refinements. These requirements are designed to ensure that the behavior of children in each refinement is equivalent to the behavior of the parent. Operation A, for example, operates on each current column without a computed difference and generates the current column with the correct difference. The nodes immediately below Operation A provide more detail as to the intermediate steps and decisions. Otherwise, however, they produce the same result. The behavior is equivalent.

In general, higher level nodes may operate on more highly structured parameters – entire columns, *CurrentColumn* for example. Corresponding lower level child nodes operate on simpler parameters, like *BorrowedDigit*. Their behavior, however, is expected to produce equivalent results.

*Assigning Instruction, Questions, Feedback and Corrective Feedback to Flexform Nodes*.-- Each node in a Flexforms represents a distinctive part of the to-be-acquired knowledge. Hence, once problem schemas have been laid out, the next step is to create and associate Questions, Instruction, Feedback and Corrective Feedback with individual nodes in the Flexforms. This easily accomplished by: a) going to each Flexform node in turn, b) calling up a dialog and c) filling in the required information associated with that node. Instruction and Questions at higher level nodes will be more inclusive (and generally less specific) than those at lower levels. In Operation 1, for example, learners would be presented with a sub-problem highlighting a column requiring borrowing (AKA regrouping). In this case, the question asked might be something like "Compute (or Find) the difference."

Questions, instructions and feedback associated with lower level nodes will typically be more specific. The Questions, Instructions and/or Feedback (positive and corrective) may include text, voice and/or visuals. TutorIT also supports a broad and extensible range of media for this purpose, ranging from built in support for text, graphics and text-to-speech plus various media support files – for example, sound (.wav), Flash (.swf) and video (.avi) files.

The sample dialog in Figure 2 shows how text, and various files may be assigned to individual nodes in the Flexform.

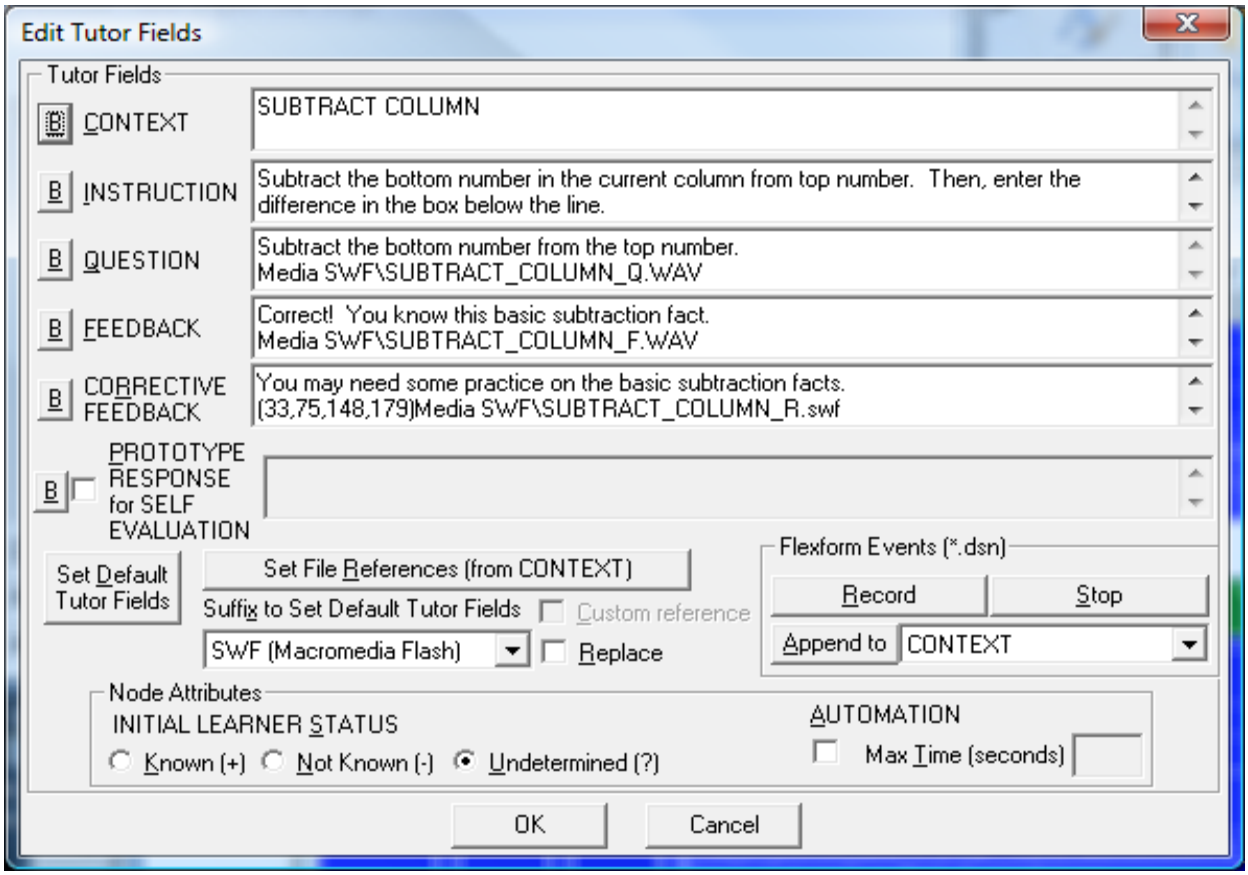



*Fig. 2 shows how text, along with sound (.WAV), Flash (.SWF) and other files may be assigned to individual nodes in the Flexform.*

*Making Flexforms Executable*.—The next step in authoring is to make the Flexform executable. Specifically, the author (or team) must implement each terminal in the Flexform so that the Flexform as a whole can be run automatically in TutorIT. This is accomplished by implementing each terminal node in the Flexform individually using AuthorIT's High Level Design (HLD) language.

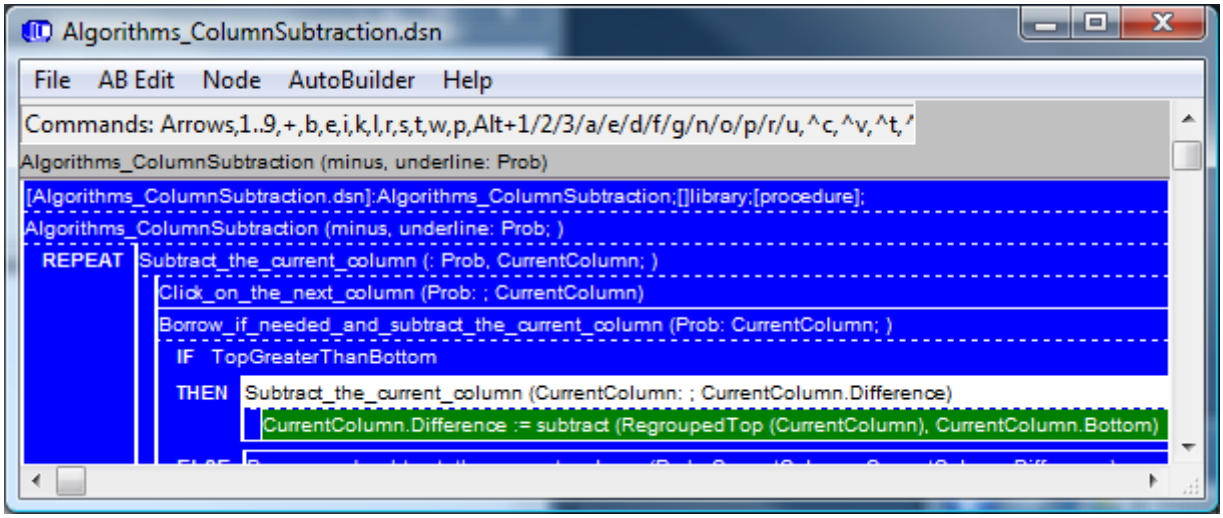



*Fig. 3. Sample High Level Design (HLD) code shown in green below the highlighted node.*

Implementing Flexforms in this manner makes it possible for TutorIT to automatically compute correct responses to any given problem or sub-problem. A simple example is shown in Figure 3. The node shown in green is the executable HLD equivalent of the highlighted node.

*Defining Problem Schemas*.-- Once the Flexform has been fully implemented (and tested using AuthorIT's build in Interpreter/Visual Debugger), the next major step is to define problem schemas that collectively exercise all nodes in the Flexform. For example, a subtraction problem with all top digits greater than or equal to the bottom ones will not exercise Flexform nodes involving regrouping (or borrowing).

Problem schemas are defined and laid out in AuthorIT's Blackboard Editor as shown in Fig. 4.

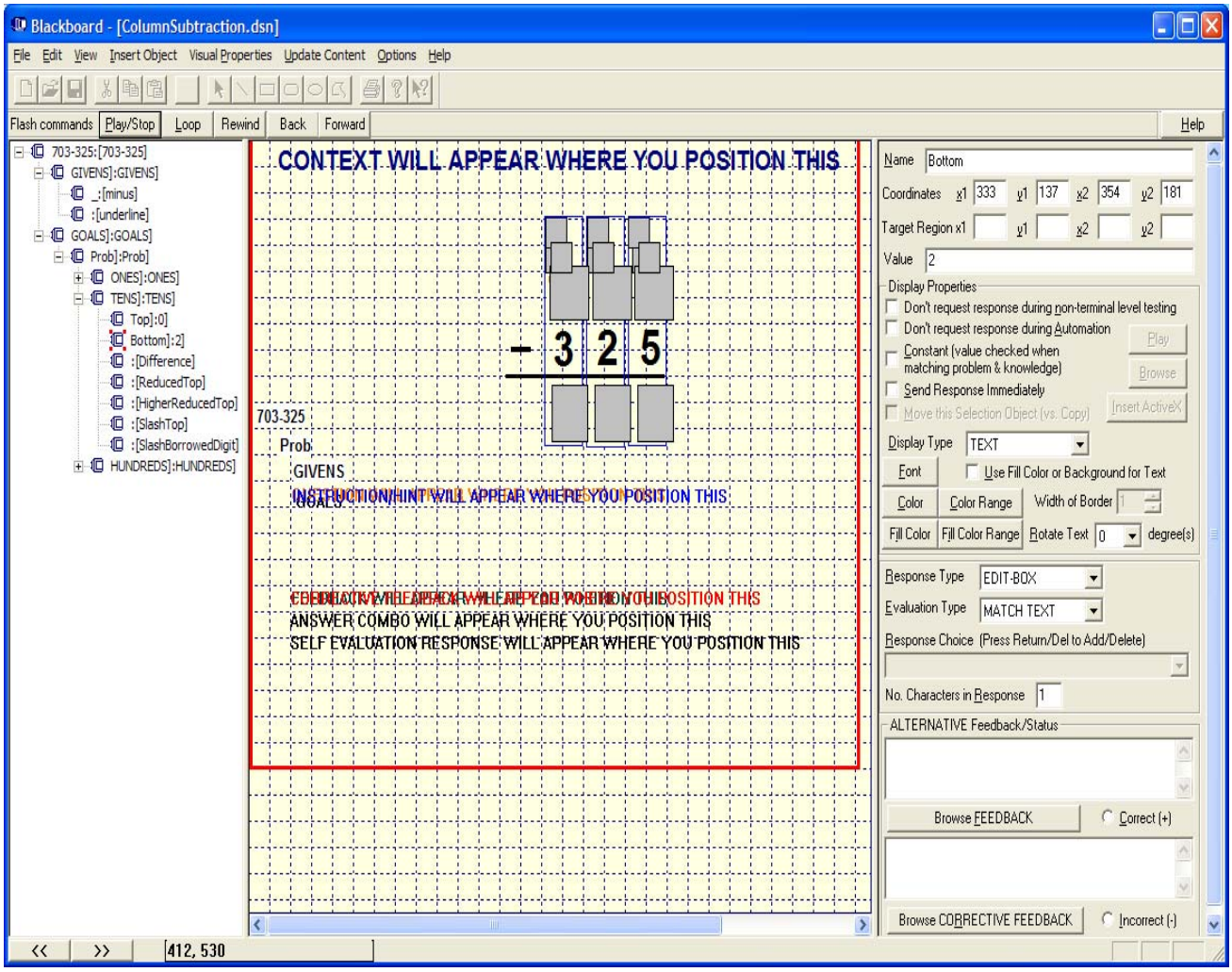



*Fig 4. The left panel in AuthorIT's Blackboard Editor (BB) Editor is used to define individual problems. The center pane is used to layout the interface through which TutorIT and the learner are to interact. The right panel is used to assign attributes to individual nodes (elements) in the problem. These attributes include Display types (e.g., Text, Flash, Animation, Sound, Picture, OLE), Response types (Edit Box, Click, Combo Box, Construction) and corresponding Evaluation types (Match_text, Within_region, Structure, Debug).*

*TutorIT Options Tool*.—The Tutor Options Tool, currently a dialog in AuthorIT, is used by authors to define/configure alternative learning modes. The first decision an author must make is to decide which of the basic TutorIT Delivery Modes to include: ADAPTIVE, INSTRUCTION, DIAGNOSTIC, SIMULATION or PRACTICE. The Options Tool in Figure 4 is set to ADAPTIVE mode. Authors also can make DIAGNOSTIC, INSTRUCTION, SIMULATION, and PRACTICE modes available in TutorIT by selecting desired modes for *TutorIT Delivery Mode* in the dialog. ALLOW LEARNER CONTROL also is an option.

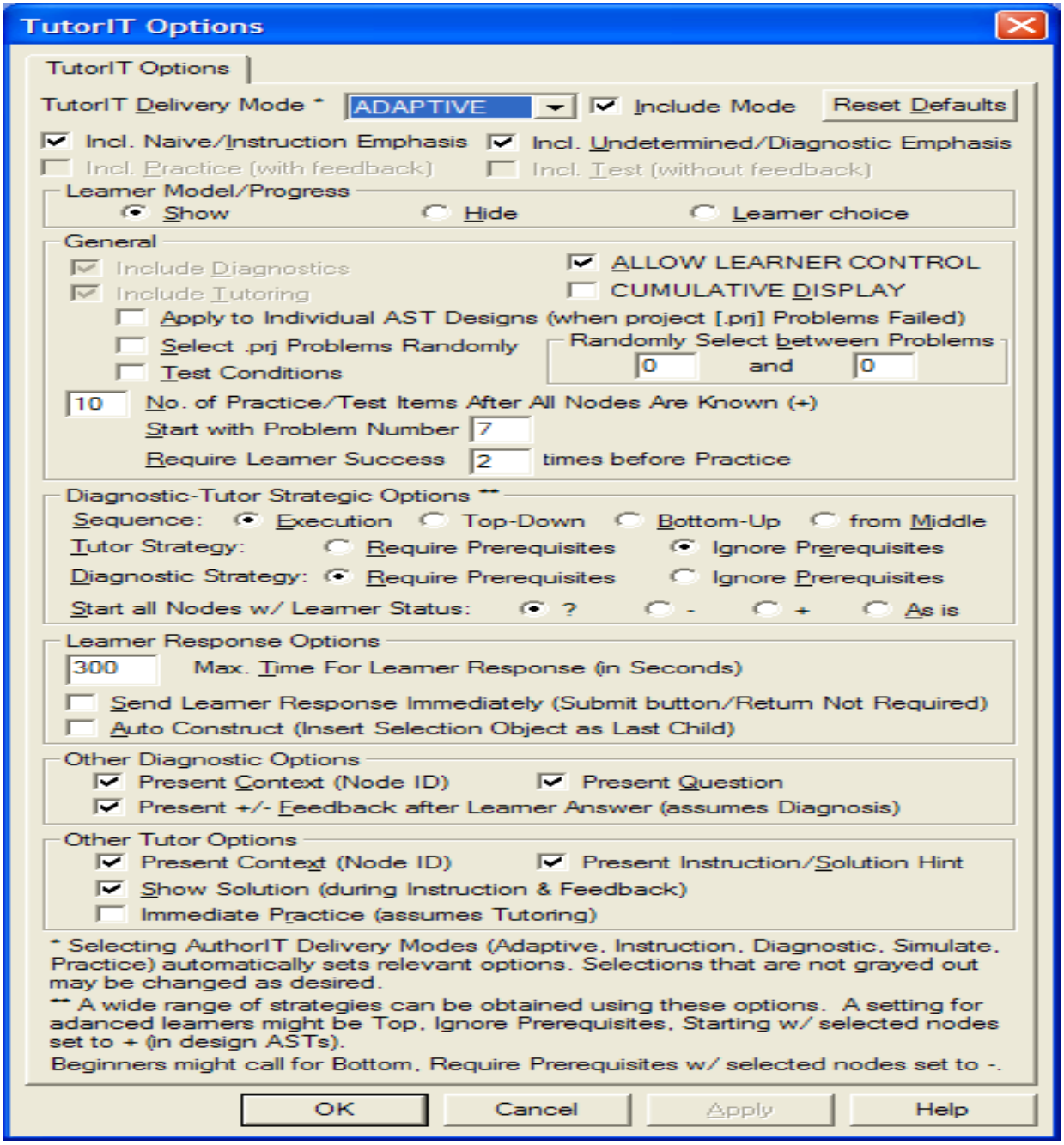



*Fig. 5. TutorIT Options Tool used by authors to define/configure alternative learning modes. Currently set to ADAPTIVE mode. Other choices allow for further customization.*

*TutorIT*.—The Flexform associated with a skill represents what is to be learned in an arbitrarily precise manner. The Flexform design (in blue) also includes HLD code (in green) which is interpretable by TutorIT. The design Flexform acts like a structured database, including all information needed by TutorIT to provide a wide variety of delivery modes. In addition to to-be-learned operations and decisions, Flexforms include: a) a modular executable implementation of each terminal (Fig. 2), b) questions, instruction, feedback and corrective feedback associated with specific nodes in the Flexform (Fig. 3),c) problem schemas (which serve as input to the Flexform) problem schemas laying out the kinds of problems to solved (Fig. 4), c) and d) TutorIT options specifying how TutorIT is to make its decisions (Fig. 5).

The way TutorIT operates depends on how it is configured in the Options Tool. I concentrate here primarily on Adaptive mode. Other options, such as Diagnostic and Instruction, are special cases or restrictions. TutorIT in Adaptive mode automatically selects nodes that quickly pinpoint what a learner does and does not know at each stage of learning.

*Learner Model*.-- TutorIT takes the above Flexform files as input and first creates a Learner Model representing what the learner initially knows or is assumed to know about the to-be-learned Skill. The Learner Model is normally displayed in a tree view with each leaf marked with a "+", "-" or "?" corresponding to a (blue or actionable) node in the Flexform (e.g., see Fig. 6).

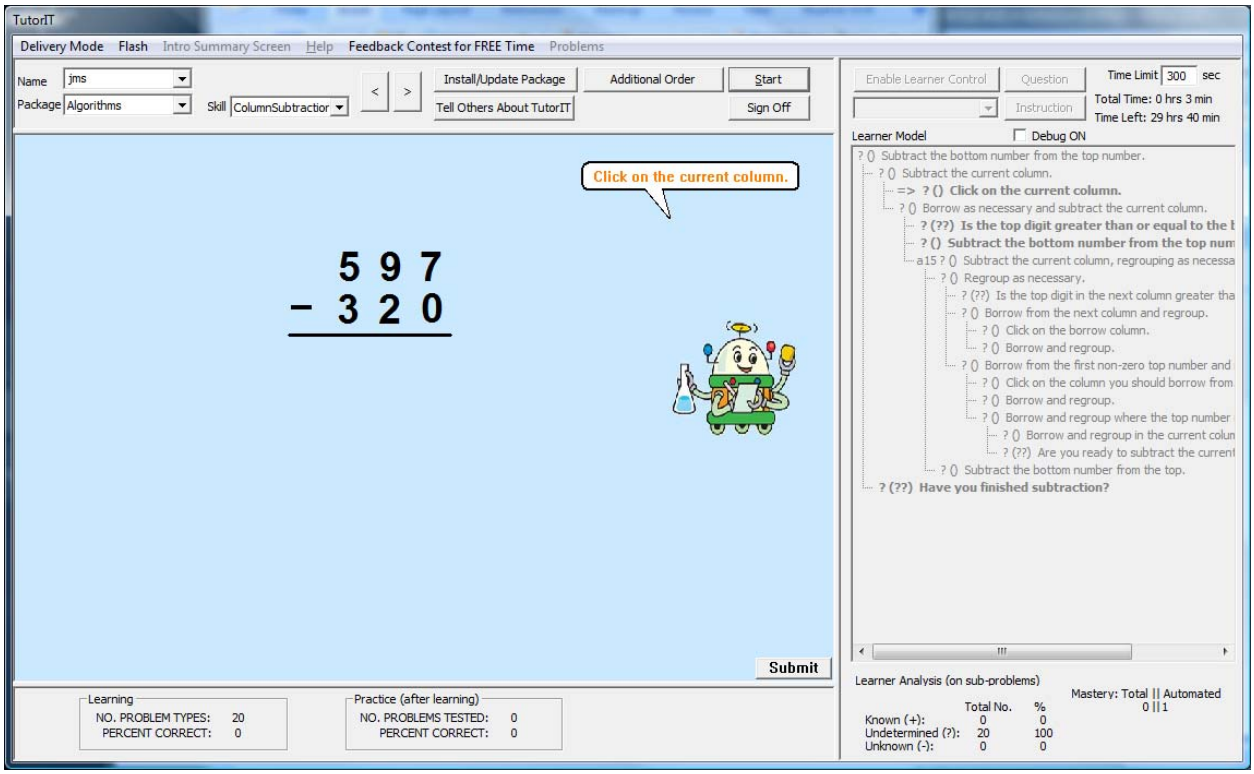



*Fig. 6. Learner Model for a student just beginning TutorIT Column Subtraction. The initial status on each node is set to "?" because TutorIT the student has had some exposure to column subtraction.*

The left side of Figure 6 shows the Blackboard interface through which TutorIT interacts with the learner. The Learner Model for a student just beginning Column Subtraction is shown on the right side of Figure 6.

Given the above information, TutorIT operates as follows. All decisions are based entirely on the structure of the content to be learned, independently of content semantics.

1. TutorIT selects a problem.
2. TutorIT then selects a (blue) node in the Flexform (or Learner Model). Only nodes that are exercised by the selected problem are eligible for selection. Selections otherwise are made according to priorities set in TutorIT's Options Tool (Fig. 5).
3. TutorIT executes the Flexform using its built in interpreter. The subtree defined by the selected node (in the Flexform) automatically generates a sub-problem of the problem schema and also its solution.

   *(NOTE: As below, we will want TutorIT to also support the case where TutorIT must generate (new) solution Flexforms from higher and lower order SLT rules. SLT's UCM will play a central role in this context. Currently, TutorIT only supports chaining two or more SLT rules as in current expert systems.).*
4. TutorIT displays the sub-problem on TutorIT's blackboard (see the left side of Fig. 6).
5. If a node is marked "-", TutorIT provides instruction. If marked "?", TutorIT presents a question to determine its status. TutorIT skips nodes marked "+" unless the node is an automation node. Automation nodes require a faster response (higher level of expertise).

   *NOTE: These questions and instructions, as well as positive and corrective feedback, may consist of simple text, voice and/or media consisting of Flash, audio-visual or other files.*
6. TutorIT compares the learner's response with the correct answer, which is automatically generated by TutorIT.
   a. If the status was "?" and the learner gets the correct answer, positive feedback is given and the node is marked correct (assigned a "+"). If incorrect, TutorIT provides corrective feedback and the node is marked with a minus ("-").
   b. If the status was "-", instruction is given and the node is marked "?". After instruction, it is impossible for TutorIT to know for sure that the learner has actually learned what was taught.

      *NOTE: The learner must meet timing requirements if the node is an automation node requiring a higher level of skill.*
7. In addition to determining the learner's status on individual nodes, TutorIT also infers what the leaner knows with respect to nodes dependent on the current one:
   a. If the learner gives an incorrect response, TutorIT reliably assumes that any (higher level) node dependent on it also should be marked unknown. TutorIT marks such nodes accordingly.
   b. Conversely, if the learner gives the correct response, TutorIT reasonably assumes that the learner also knows those lower level nodes on which it depends. In short, TutorIT not only compares learner responses on sub-

problems corresponding to individual nodes but also quickly infers what the learner knows about nodes dependent on it.

To date, we have developed TutorIT tutorials for Column Addition, Column Subtraction, Column Multiplication and Long and Short Division along with five levels for each of the Basic Facts: Addition, Subtraction, Multiplication and Division. Operations on Fractions also are in progress. Please see www.TutorITmath.com for latest availabilities.

In each case, TutorIT takes problem schemas as laid out in the Blackboard Editor as input. It automatically generates problems, actually sub-problems, as needed for diagnosis and remediation. Nodes are selected so as to enable TutorIT to quickly pinpoint what each individual does and does not know at each point in time, and to provide precisely the information (instruction) needed when needed to progress in optimal fashion.

All this is done dynamically during the course of instruction as might a human tutor. The main difference is that TutorIT does this in a highly disciplined manner. All decision making is done automatically based entirely on the structure of the to-be-acquired knowledge. Semantic independence dramatically reduces the effort required to create adaptive tutoring systems.

The hierarchical representation of knowledge (in Flexforms) has important implications for both efficiency and effectiveness. As above, TutorIT makes direct inferences not only with respect to individual nodes but to dependent nodes as well. For example, if a student gets a problem associated with one node correct, then TutorIT can reliably assume that the student also knows all of the lower level nodes on which it depends. For example, if a child can subtract columns involving borrowing or regrouping, one can reasonably assume that the child can also subtract successfully when there is no regrouping. On the other hand, if a child cannot subtract a column that does not involve regrouping, one can be quite certain, he or she cannot subtract when regrouping is required. In short, success on a node implies success on all subordinate nodes. Failure implies failure on all superordinate nodes. The result is very efficient diagnosis and instruction.

Unlike most teachers, TutorIT can be unusually effective because it benefits from careful pre-analysis. We have put a considerably amount of effort into our TutorIT Math skill tutorials – far more than what goes into writing a text book for example. The level of analysis in our current prototypes can and will be further improved as a result of field testing. Even in their current state, however, TutorIT Math skill tutors benefit from considerably more analysis than most teachers are capable. And, this analysis can further be improved incrementally.

*Importance of Prerequisites*.— One might argue that just because a student solve one subproblem (associated with a given node) does not necessarily imply that he or she can do this with all such subproblems. Indeed, this is correct. As pointed out in my earlier description of TutorIT (Scandura, 2005) a single test will only be sufficient when analysis is complete – when all terminal nodes are as we say atomic. Success on one instance in this case implies success on all instances of the same type with unusually high degrees of reliability (cf. Scandura, 1971a, 1973, 1977). Having said, this just as a good bridge designer builds in a safety factor, an author can easily do the same with TutorIT. TutorIT can be required to demand a higher level of performance by simply changing a setting in the Options Tool to require any number of successes on each node (before mastery is assumed).

Criteria may be set so as to actually guarantee learning. Any learner who enters with pre-specified prerequisites, and who completes a given TutorIT tutorial will be definition have

mastered the skill in question. There is no other way a student can complete a TutorIT tutorial. He or she must meet pre-specified criteria set by the author or TutorIT tutoring will continue until they are met.

Notice that prerequisites play an essential role in the process. Prerequisites correspond precisely to atomic or terminal nodes in Flexform knowledge representations. Some prerequisites are so simple that they can safely be assumed on entry. For example, the ability to read and write numerals. Entry with respect to other prerequisites, however, may be less certain. Any child presumed to be ready for long division would almost certainly have to know the multiplication tables and how to subtract. Similarly, no would want to teach column subtraction unless a child already knew how to count.

The basic question in this context is how one make contact with learner's who have not mastered such prerequisites? For example, how to teach column subtraction to a child who cannot write or recognize numerals (e.g., "5", "3"). SLT support for indefinite refinement offers a unique solution to this problem. One is not forced to introduce non-decomposable relationships. Instead, each such prerequisite can be represented as an equivalent SLT rule with its own domain and range. As above, for example, the numeral "5" can be viewed as an SLT rule for constructing the numeral from more basic line segments. Most important, SLT rules representing prerequisites can be refined further just as any other.

One further point. Let's turn this argument on its head. The difficulty of any task, or to be learned skill depends not on just the skill itself. Rather, it depends on the nature of the prerequisites that may be assumed available. We hear a lot, for example, about the benefits of using sophisticated calculators in education (e.g., TI's Nspire family). Clearly, if one has a calculator, computational issues take a back seat. It is far easier to learn to evaluate arithmetic computations with a calculator than without.

*NOTE: Along with most mathematics educators I would argue nonetheless that computational abilities are essential irrespective of the presence or absence of a calculator.*

On the one hand mastering Nspire can be can subjected to the same kind of analysis we are talking about here. And, TutorIT could equally well be used to provide the necessary instruction. On the other side of the coin, one can start with the assumption that learners can already use of such tools as Nspire – as prerequisites on entry. In this context, to-be-analyzed problem domains will be very different.

Instead of computational skills, the focus is more likely to be on problem analysis. Given a description of a situation, for example, how can it be formulated in terms of mathematical expressions? Having created such an expression, one can plug in the numbers and click to get the solution. In a similar manner, the more comprehensive the skills one can assume the more sophisticated the knowledge one can teach. The general truism to be taken from this analysis is not whether basic skills are important but rather that the more basic skills one has mastered, the more one has to build one. This is true whether in mathematics or in any other subject.

*NOTE: Representing reality in terms of mathematical expressions is one of six basic process abilities in mathematics. These were first introduced in Chapter 1 of my book on Mathematics: Concrete Behavioral Foundations (Scandura, 1971b, pp. 3-64). The six abilities were organized as three bidirectional pairs: Detecting regularities and its opposite of constructing examples of regularities, understanding mathematical representations (e.g., expressions) and its opposite of creating mathematical expressions and deduction and its opposite axiomatization.*

*Configuring TutorIT*.-- The ease with which TutorIT can be customized adds another important dimension. In addition to ADAPTIVE mode, the Options Tool also supports

DIAGNOSTIC, INSTRUCTION, SIMULATION and PRACTICE modes. Authors may also allow learner Control, in which case the learner may decide on which items to be questioned or to receive instruction.

Each basic delivery mode comes with some mandatory settings. Other options enable authors to better control the way content is delivered.

At the most basic level, for example, a student might already have been exposed in varying degrees to the knowledge being taught. In this case, TutorIT cannot know what the learner knows on entry. In so far as TutorIT is concerned, the learner enters essentially as a blank slate. Conversely, if a student has had no exposure to the content, TutorIT might start with nodes marked "-""-", or unknown. In this case, TutorIT will initially be biased toward instruction.

In this case, "*Start all Nodes with the Learner Status*" in AuthorIT's Options Tool (see Fig. 4) might be set to "?" or "-" depending on prior student exposure to the content,. The author also has the choice of allowing teachers or students to make the choice for individual students or to require it for all.

In the undetermined state, TutorIT starts tutoring each child by marking each node in the Learner Model (see below) with a "?". This signifies that TutorIT does not (yet) know whether or not a learner has mastered the knowledge associated with that node. After the learner responds, TutorIT provides corrective or positive feedback as appropriate – and updates the Learner Model as above -- "+" for success or "-" for failure.

Other options provide finer levels control. For example, an author might require that instruction be given only when ALL prerequisite nodes have been mastered (marked "+"). Alternatively, the author might want to place more emphasis on self-discovery. Here, the author might choose the *Ignore Prerequisites* option for *Tutor Strategy*. In this case TutorIT will provide hints/scaffolding (i.e., instruction) even when the learner's status on lower level nodes is unknown.

More generally, the author has a wide variety of options making it possible to accommodate a wide variety of pedagogical biases – or should I say "instructional theories". Available options support a wide variety of instructional philosophies -- ranging from highly directive instruction to open ended discovery including completely self directed learning.

*Comparison and Benefits*.-- Like other ITS or CBI, TutorIT Math tutors are highly reliable. They never tire. They never make mistakes – excepting bugs one may have missed. Unlike other CBI (or ITS), however, TutorIT Math tutors are designed so that any learner who enters with pre-specified prerequisites and who completes a given tutorial will necessarily have mastered the skill in question.

Whether these results are realized with actual students depends on the following assumptions: a) that we have in fact identified an SLT rule for correctly performing a ranges of basic math skills with sufficient precision to have identified essential prerequisite skills (terminals in Flexform used to represent SLT rules), b) that learners demonstrate mastery of those prerequisites on entry and c) that students complete the TutorIT tutorial – the only way a student can do this is to have demonstrated mastery on the skill being taught to whatever criterion the author has prescribed (in the Options Tool).

In effect, what the student learns and whether or not a student who completes a tutorial actually learns the skill is not a question to be determined empirically. Rather, the proof will be in such

things as how long it takes, whether students are sufficiently motivated to complete the tutorial, and generally what might be done to make the tutorial even better (e.g., more efficient and/or motivating for students, etc.). Given the way TutorIT tutorials are developed, improvement will occur incrementally as feedback suggests and as resources allow.

Toward this end, we are just now beginning field testing by offering free trials. Anyone, a school, tutoring center, or home can get free individualized tutoring on whatever skills are currently available (now 5 levels for reach of the basic facts and the basic whole number algorithms for addition, subtraction, multiplication and division).

At the same time, the AuthorIT/TutorIT system dramatically reduces development costs. As above, we have already developed a range of TutorIT tutorials at a fraction of the costs of ITS development. These tutorials focus on very specific identifiable skills. Guaranteed learning is restricted specifically to those skills. Nonetheless, TutorIT tutorials developed to date also include instruction pertaining to meaning. I refer here to the kinds of instruction commonly included in textbooks and classroom instruction.

There is no guarantee having gone through a given TutorIT tutorial that students will necessarily also master this supplemental material – material that is normally included in classroom instruction. The question of whether and to what extent this supplemental instruction benefits students is an empirical one. Given TutorIT's focus on doing what it can do better and more efficiently than a human (or any other means of transmittal), this question also is of secondary importance. Current TutorIT tutorials are designed to support classroom instruction and textbook learning not to replace them.

How can that be – better results at lower cost? The answer lies in the very close relationship between knowledge representation, on the one hand, and diagnostic and remedial actions on the other. On the one hand, arbitrary refinement allows for indefinite precision. Tutoring can be guaranteed. Learners who enter with predetermined prerequisites and who complete a given TutorIT tutorial will by definition demonstrate mastery of defined skills.

The same structural relationships that make it possible to provide efficient, highly targeted adaptive instruction also eliminate the need to program pedagogical decisions. While estimates may be based on slightly different assumptions, the bottom line is that roughly half of all development costs can be eliminated (cf. Foshay & Preese, 2005, 2006; Scandura, 2006a,b). It is not necessary to independently program diagnostic and instructional logic as in developing other adaptive tutoring systems.

In comparison with other approaches, AuthorIT and TutorIT offer three major benefits:

a) Better results on well defined tasks than even human tutors due both to more complete analysis (than most humans are capable of) and to highly effective and efficient tutoring. The latter derives from TutorIT's optimized pedagogical decision making. More complete analysis and optimized decision make it possible under carefully prescribed conditions to actually guarantee learning.

b) Greatly reduced development costs because all TutorIT decision making is predefined. All diagnosis and testing is automatic and based entirely on the structure of the to-be-learned knowledge. While we have not kept actual figures on development costs, they are by definition an order of magnitude less than that required in ITS development. TutorIT tutorials ready for field testing have been completed by myself with the assistance of approximately one full time person for a year. With an experienced team

and further maturity of AuthorIT and TutorIT, development costs may be expected to go down gradually.

c) Furthermore, pedagogical decision making is fully configurable. TutorIT can easily be configured to provide adaptive tutoring customized for different learners both individually and by population. TutorIT also can be configured to provide highly adaptive diagnosis, to provide practice or to serve as a performance aid. Configuration consists entirely of making selections in an Options dialog – all without any programming or change in the knowledge representation.

*NOTE: The notion of (content) domain independent instructional systems is not entirely new. It is not difficult, for example, to construct CBI systems that support specific categories of learning, such as those defined by Gagne (1985). The closest analog is probably Xaida (e.g., see Dijkstra, Schott, Seel & Tennyson,1997). TutorIT takes a major step forward in this regard by providing tutoring support for ANY well defined content. This not only includes all Gagne's categories of learning, for example, but any combination thereof.*

The bottom line is that TutorIT is another significant step forward in automation. TutorIT provides another case where computers can do things better than a human -- this time in adaptive tutoring. As more and more tutorials are developed TutorIT can gradually taking over tasks previously done by humans – not just in math skills but ultimately with any well defined skill. TutorIT tutorials will gradually take over for one reason: Not because they are approaching what humans can do but because they can do some jobs better than humans.

TutorIT, of course, will not eliminate the need for good teachers any more that good computational tools have eliminated the need for people who use them. TutorIT tutorials will enable teachers to concentrate on things they can do better. TutorIT automation will be an on-going and continuing process. Our children and our country will be the main beneficiaries. If you might be interested in contributing to this effort please let me know at scandura@scandura.com.

## 4. Critical Advances in Current SLT Theory

*Analyzing Complex Domains*.— It might seem we are done! Given any domain, we can always use AuthorIT's AutoBuilder component to systematically identify what needs to be learned for success – with whatever degree of precision may be necessary or desired. The rest follows automatically. TutorIT takes the representation produced (including display layouts and associated media) as input and automatically delivers instruction as prescribed.

While theoretically possible, identifying what must be learned as an SLT rule is not necessarily easy. It can be very difficult, practically impossible to identify a single, integrated SLT rule that represents the knowledge needed to master complex domains. This is not simply having to compromise as regards completeness.

It would be impractical if not impossible to directly identify what must be learned to prove all known theorems in mathematics, or to specify how to write a beautiful poem (given some topic or idea). As those engaged in ITS development know, identifying what needs to be learned in high school algebra already poses a difficult task (Ritter, 2005).

By *way* of contrast, high level relational models are relatively easy to create (cf., Scandura, 1973; Hoz, 2008). Relational models, however, lack precision -- and complexity increases rapidly. Both constraints place significant limits on effective tutoring. Equally important, pedagogical decisions based on relational models can be very difficult. Pedagogical decision making depends inextricably on content semantics, thereby increasing both development and evaluation costs.

In SLT, it might appear that one can avoid this problem by simply introducing a finite set of SLT rules. For example, instead of one SLT rule as above, why not simply add new SLT rules? Certainly, one can do this. Doing so, however, does not solve the fundamental problem. Given any non-trivial domain, it is impossible to directly identity everything a learner should know. This fact has been a central tenet in SLT from its inceptions (cf. Scandura, 1971a). It was the primary motivation for introducing higher order rules.

ITS systems approach this problem from a very different perspective. Beginning with Newell & Simon's (1972) influential work on problem solving, the focus has been on identifying sets of productions corresponding to what might be in human brains. ITS knowledge engineers work with subject matter experts to identify condition-action pairs, or productions representing relevant knowledge. Productions collectively are expected to be sufficient for solving arbitrary problems in a given domain.

Identifying productions, however, is not sufficient in itself. Give a computer a problem and a set of productions, and what happens? Nothing! As Newell & Simon (1972) recognized early on some kind of control mechanism is necessary to activate the productions.

From a theoretical perspective the fewer mechanisms needed the better. With this in mind, Newell & Simon (1972) originally proposed "means-ends" analysis as a universal control mechanism: Given a problem, select (and apply) productions that will reduce the difference between the goal and the current problem state. Mean-ends analysis seemed reasonable and gradually morphed into chaining (of productions). Empirical results later suggested that other mechanisms also are commonly involved in learning and problem solving: Variations on generalization, abstraction, analogy and other mechanisms have been proposed.

A further limitation of learning mechanisms, as used in production systems, is that they impose essential constraints on implementation. One can add or remove individual productions without fundamentally changing the operation of an ITS. Learning mechanisms, however, necessarily

come "hard wired". They cannot be added or removed without fundamentally effecting operation of a production system.

Ohlsson (2009) suggested no end to the number of learning mechanisms that might be needed or desired. The impracticability of identifying all potentially relevant mechanisms is one of the reasons that he and Mitrovic introduced Constraint Based Modeling as a means of reducing complexity in ITS development (e.g., Mitrovic & Ohlsson, 2007).

Quite independently, Polya's (1960) early analyses of mathematical problem solving further suggest that learning mechanisms are, in fact, domain dependent. Polya identified a number of domain specific "heuristics" like the pattern of "two-loci" or "similar figures". Such heuristics are formally equivalent to learning mechanisms, but are more similar in nature to higher order rules in SLT (cf. Ehrenpreis & Scandura, 1974; Wulfeck & Scandura, Chapter 14 in Scandura, 1977). Higher order SLT rules are domain dependent and play a direct role in how new SLT rules are acquired and used.

*NOTE: While influential in mathematics education, Polya's (1960) work is not widely known in TICL circles.*

*SLT Solutions*.—SLT takes this analysis further. Existing SLT theory offers a detailed road map going forward, a road map that builds directly on current AuthorIT and TutorIT technologies.

Consider the second or third major advances mentioned earlier:

**(2) The ability to systematically identify the higher as well as lower order SLT rules required for success in any given domain, no matter how complex.**

**(3) The ability to formulate SLT's Universal Control Mechanism (UCM) in a way that is completely independent of the rules and higher order rules necessary for success in any given domain.**

I summarize each of these advances and their importance below. Then, I describe how AuthorIT and TutorIT can be extended to support each advance.

*Structural (cognitive domain) Analysis (SA) of Complex Domains*.-- SA takes a fundamentally different approach to the problem. The focus here is on identifying both the higher and lower order SLT rules that must be learned for success. Unlike productions (condition-action pairs), SLT rules are not assumed to be in human minds – nor are higher order rules viewed as hard wired mechanisms. Rather, higher as well as lower order SLT rules (like relational models) are both operationally defined in terms of observable behavior with respect to criterion tasks.

All SLT rules represent what must be learned for success. They provide an explicit basis for both diagnosis and remediation.

Historically, Structural (cognitive domain) Analysis (SA) has been used to systematically identify higher as well as lower order SLT rules. As detailed above, the use of ASTs to replace directed graphs has played an important role enabling automation in the development and delivery of adaptive tutoring systems (cf. Scandura, 1971a, 1973, 1977 where SLT rules are represented as directed graphs or flowcharts and Scandura, 2005, 2007 where SLT rules are represented in terms of ASTs). The process by which higher order SLT rules were constructed, however, was largely subjective.

The way higher order SLT rules were constructed was fine for paper and pencil courseware development (e.g., a workbook by Scandura et al, 1971c) and for experimental research (e.g., 1974a). But it was not sufficiently systematic or precise for automation.

As SA was originally defined, the analyst, typically but not necessarily a subject matter expert or instructional designer, was asked to:

a. define a complex problem domain informally,

b. select a finite set of prototypic problems in that domain,

c. construct an SLT solution rule for solving each prototype problem,

d. construct a higher order SLT rule operating on other SLT rules (for constructing each solution rule),

e. eliminate redundant SLT rules (which can be derived by application of higher order rules to others), and

f. repeat the process as desired, each time resulting in a set of SLT rules that were at once simpler and collectively more powerful in generating power.

SA was continued until the SLT rules and higher order rules identified provide sufficient coverage of the domain (cf. Scandura et al 1974 and Wulfeck & Scandura, 1977).

Analysis of various complex domains (e.g., Scandura et al, 1974, Scandura, 1977, Scandura & Scandura, 1980) shows that as SA proceeds two things happen: The individual rules become simpler but the generating power of the rule set as a whole goes up dramatically, thereby expanding coverage in original domain (esp. see Scandura, et al, 1977; Wulfeck & Scandura, 1977).

*NOTE: There is no loss of generality because domains can be incrementally expanded without loss by building successively on prior analyses. For details, see Scandura (2007) for the most complete coverage of the basic theory.*

Choosing the appropriate level of analysis in Step c was originally ad hoc. This difficulty was solved as above by the introduction of ASTs. Each individual SLT rule can now be refined successively in whatever degree of precision may be necessary or desirable. .

Step d of constructing higher order rules, however, was still a bottle neck – too subjective for full automation. The key to solution was the following missing link between steps c and d:

Convert each SLT solution rule in Step c into a higher order problem.

Once a higher order problem has been constructed, higher order SLT rules can be constructed in exactly the same way as all other SLT rules.

Given any problem domain, no matter how complex, the goal of Structural Analysis is to identify a finite set of higher and lower order SLT rules – rules that collectively make it possible to solve a sufficiently broad range of problems in the domain. Unlike production systems, where the focus is on identifying ingredients that might be in human brains, the focus in Structural Analysis is on identifying what must be learned for success.

Consider the following example of SA applied to a Number Series domain (adapted from Example 3 in Scandura (2007). I have selected this example because it illustrates not only higher order SLT rules that generate new SLT solution rules but also how higher order selection rules come into play. (See Appendix A for other examples.).

**Number Series Domain**

**1. SME Selects Prototypic Problems (one of potentially many)**

*1 + 3 + 5* -- *?sum*

**2. Construct (multiple) Solution Rules (2A, 2B, 2C) for Prototypic Problem**

| | | | |
|---|---|---|---|
| *2A* | *1 + 3 + 5* | *→3x3→* | *9* |
| *2B* | *1 + 3 + 5* | *→3x(1+5)/2→* | *9* |
| *2C* | *1 + 3 + 5* | *→successive addition→* | *9* |

**3. Convert SLT Rule to Higher Order Problem**
**(Construct Goal & Given of Higher Order Problem)**

**Higher Order Problem 3A:**

| | | |
|---|---|---|
| *1 + 3 + 5* | *→3x3→* | *9* |
| *1 + 3 + 5 + 7 + … →nxn→* | | *Sum* |

**Higher Order Problem 3B:**

| | | |
|---|---|---|
| *1 + 3 + 5* | *→3(1+5)/2→* | *9* |
| *a + a+d + … + L=a+(n-1)d* | *→n(a+L)/2→* | *Sum* |

**Higher Order Problem 3C:**

| | | |
|---|---|---|
| *1 + 3 + 5* | *→1+3+5→* | *9* |
| $a_1 + a_2 + a_3 + \ldots + a_n$ | *→successive addition→* | *Sum* |

**4. Alternative SLT Higher Order Rules for Solving Higher Order Problems 3A, 3B, 3C**

| | |
|---|---|
| **Higher Order Rule 3A:** | **→replace 3 terms by n→** |
| **Higher Order Rule 3B:** | **→replace 1 by a, 5 by l, 3 terms by n→** |
| **Higher Order Rule 3C:** | **→replace each term by a variable, three terms by n→** |

Notice that each alternative higher order SLT rule has a different domain of applicability. Higher order rule 3A is very efficient but only works with arithmetic series beginning with 1 and having a common difference of 2 -- for example, *1 + 3 + 5 + … + 99 →50x50→ 2500.* Rule 3B is reasonably efficient and works with all arithmetic series. Rule 3C is relatively inefficient (especially with long series) but works with all number series, arithmetic or otherwise.

*(NOTE: For early empirical research on the subject see Scandura, Woodward & Lee 1967; Scandura 1967.)*

In effect, three higher order rules are applicable rules in this example. Deciding which one to use is essentially what one must do in many design problems. The acquisition of multiple ways of solving any given problem and of knowing which to select when is a key characteristic of expertise.

In our example, the selection process represents a still higher order problem. The given in the problem consists of the three alternative rules. The goal is to select exactly 1. One higher order SLT selection rule that works can be summarized as:

**Case Type-of- Number Series:**

**a) Starts with 1 with a common difference of 2 → select rule $N^2$**
**b) Common difference → select rule N(A+L)/2**
**c) Else →select successive addition**

A more general but error-prone selection rule is to simply choose the simplest rule. Domain of applicability was largely ignored in early research. Defining the domain structures associated with higher order SLT rules is essential. Automatically perceived structures play a decisive role in determining which rules to use under what circumstances.

*THEORETICAL NOTE FOR THOSE INTERESTED IN TRAINING EXPERTISE: For those who have read my recent monograph (Scandura, 2007) I would like to add one general remark: In that monograph I introduced the notion of higher order SLT automation rules as the mechanism by which more efficient (automated) rules are derived from other rules. Irrespective of how they are learned I suspect that most expertise is gained via the gradual acquisition of efficient, increasingly specialized solution rules. Apparently effortless expert behavior results when previously learned, more efficient SLT rules are selected (via higher order selection rules) for use in more and more situations. In accordance with SLT's Universal Control Mechanism (UCM), these more efficient rules are selected by applying higher order (selection) rules as in all other behavior. The result is increasingly efficient, apparently automated behavior.*

*The Need for Learning (often called Control) Mechanisms*.— All knowledge in SLT is strictly relative: What a person knows is defined by that person's behavior relative to what must be learned for success. This relativistic view of knowledge holds whether the knowledge in question is of a higher or lower order. Whereas lower order SLT rules correspond to productions in expert systems, higher order SLT rules correspond to learning mechanisms.

The question then is what controls the use of SLT rules? History makes it clear that neither means-ends analysis, chaining, nor any other expert system mechanism is sufficient. Furthermore, experience with Structural Analysis makes two things clear

a) All mechanisms that have been proposed MAY play a role in problem solving and

b) Variations on all such rules can systematically be derived via Structural Analysis.

Any automated system capable of solving problems must include some kind of control mechanism. The system must know what SLT rule to use and when. I first proposed Goal Switching for this purpose in an invited talk where I first introduced SLT at AERA in 1970 (published in Scandura, 1971a). Unlike chaining and the like, SLT's goal switching was originally modeled on a very easy to state but very hard to implement truism: *Given a problem for which no solution is immediately available, the problem solver must necessarily first derive a procedure for solving the problem*. Indeed, this truism was so general, and so commonsensical that it took considerable convincing to get supporting experimental research on UCM published in the traditionally very rigorous Journal of Experimental Psychology (Scandura, 1974a).

Goal Switching obviously differed from Newell & Simon's (1972) means-ends analysis. Indeed, Newell served as a reviewer and proposed rejecting several of my articles during this time period, including to one above in the Journal of Experimental Psychology (Scandura 1974a) and another in Artificial Intelligence (Scandura et al, 1974). Fortunately, my counter arguments and other reviews led to their eventual publication.

In fact, however, a major limitation of Goal Switching had nothing to do with validity or relevance. A series of formal experiments (Scandura, 1967), as well as more informal pilot research with subjects as young as 4 years old, demonstrated its (near) universal availability to all learners. The difficulty was in attempts to formally implement Goal Switching in a way that was completely independent of ANY higher order rule (cf. Wulfeck & Scandura, 1977). This was finally accomplished with formalization of SLT's Universal Control Mechanism (UCM) in the early 2000s (see Scandura, 2007, U.S. Patent, 6,275,976). Again, I won't repeat here what is already in print (see Scandura, 2007, for specifics).

In retrospect, one can see why expert systems run into trouble. The original hope was that there were only a small number of basic learning mechanisms – preferably one. Alas, "means-ends analysis" as originally proposed by Newell & Simon (1972) turned out not to be that mechanism.

SLT's Universal Control Mechanism (UCM), on the other hand, serves this role in unique fashion (Scandura, 2007; cf. Scandura, 1971a, 1973, 1974a,b). UCM is completely independent of SLT rules and higher order rules. More important, and unlike means-ends analysis, chaining and other mechanisms proposed in the expert system world, UCM serves as a common denominator completely independent of any particular problem domain.

An overview of UCM follows (for details see Scandura 2007.):

- Check available rules to see which SLT rules have structures that match the given problem
- Unless exactly one SLT Rule matches, control goes to a deeper level looking for rules whose ranges contain structures that match the given problem (a recursive process)
- Once exactly one SLT rule is found, that rule is applied & a new rule generated
- Control reverts to the previous level & the process continues with checking at the previous level of embedding
- Eventually, the process halts because the problem is solved or processing capacity is exceeded (alternatively a predetermined recursion limit may be set in automated systems)

Measuring knowledge relative to behavior in one form or another is not new. However, being able to explain and predict the behavior of individuals in specific instances distinguishes SLT. This is true even more so where a problem solver does not already know a solution procedure -- but must derive one. UCM plays an essential role in the latter process.

*NOTE: A historical analogy to Relativity Theory is interesting in this regard. Without assigning more significance than warranted, introduction of UCM in SLT plays a role analogous to constancy of the speed of light in Relativity Theory.*

*Measuring speed of an object relative to an observer was not especially new or interesting. Add in the constant speed of light, however, and the situation changes. As Einstein showed in 1905 funny things happen when one accounts for the time light takes to reach an observer.*

*Knowledge is strictly relative in a similar sense. What counts as knowledge is not absolute but necessarily relative to observable behavior.*

*Behavior with respect to complex domains may be explained via finite sets of higher and lower order SLT rules. But, these SLT rules depend on analyst.*

*UCM is what holds things together. Together with the SLT rules and higher order rules associated with any given domain, UCM allows explicit predications regarding problem solving behavior in specific instances (Scandura, 1974a).*

## 5. Needed AuthorIT and TutorIT Extensions

To date, AuthorIT/TutorIT tutorials have only been used to develop tutorials for well-defined math skills. TutorIT does, however, support "chaining" although this technologies has not been put to serious use. The only example to date involves a simple railroad crossing, where TutorIT is fed two simple rules: a) one for turning a signal red or green depending on the location of a train (near or out of a crossing) and b) one for raising or lowering a railroad crossing gate depending on the color of the signal (red-down and green-up). TutorIT is not explicitly told that the gate must go down when the train approaches the crossing and up when it is not.

TutorIT is able to generate correct answers by chaining known rules, where the output of one serves as input to the next as required to generate the correct answer. The answers TutorIT generates are used in turn to evaluate learner responses. Chaining is a small step forward and akin to what is done in contemporary ITS systems.

As outlined above (and detailed in Scandura, 2007; Scandura et al, 2009), however, SLT goes much further. Two objectives are on the near term agenda.

1. In order to teach higher order SLT rules we must be able to systematically identify and precisely represent them. The behavioral equivalent of all other learning mechanisms that have been proposed or used in Intelligent Tutoring Systems (ITS). or expert systems generally, can be represented as higher order SLT rules. These higher order SLT rules are derived directly via Structural (domain) Analysis (SA) from the problem domain itself.

2. In order for TutorIT to generate solutions to ill-defined problems, we must also be able to formalize and implement SLT's Universal Control Mechanism (UCM).

Both AuthorIT and TutorIT will both have to be extended. First, AuthorIT must support the construction of SLT rules that operate on nodes that are themselves SLT rules.

This can already be done using AuthorIT's SoftBuilder component. SoftBuilder is a fully general development system. It supports the construction of any kind of SLT rule. Any SLT rule, whether of a higher or lower order, can be represented as a Flexform. While sufficient in principle, however, it is extremely complex to construct higher order SLT rules. The basic task is hard enough. But, there is no automated support for refining higher order operations (or data) as is currently the case with AutoBuilder.

SA in SLT does provide the necessary rigor. The major work needed is to add support for what are called dynamic structural refinements and corresponding interaction procedural refinements (e.g., see Scandura, 2007, esp. pp. 195-198). As detailed on pages 194-216, Structural Analysis so extended would make it possible to construct arbitrary higher order SLT rules as needed.

The second major improvement requires replacing TutorIT's current chaining mechanism with SLT's Universal Control Mechanism (UCM). Fortunately, the chaining mechanism is a separable module so its replacement and integration should be straight forward. Furthermore, the UCM design has been detailed in a recent patent. The main challenge is to implement, test and refine as necessary to ensure that all work as designed ready for prime time.

I will not than attempt to detail here either the extended form of Structural Analysis or the UCM (Scandura, 2007), and I certainly don't want to imply that this will be a trivial undertaking. The risks are high. For details I encourage you to study my recent monograph (Scandura, 2007, for SA – esp. pp.216-231 and UCM – esp. 216-231).

What is important here is to understand that extension of AuthorIT and TutorIT will do two major things for us:

1) AuthorIT's AutoBuilder component will fully support Structural (domain) Analysis (SA), enabling it to identify and detail higher as well as lower order SLT rules associated with any given domain.

2) TutorIT enhanced with UCM will be able to solve novel problems in domains, even where it is not explicitly given a SLT solution rule.

Given a complex domain, extension of AutoBuilder will more fully support Structural (domain) Analysis (SA). In addition to arbitrary refinement, AutoBuilder will be able to systematically identify finite but sufficient sets of higher as well as lower order SLT rules. Sufficiency means that collectively these SLT rules will provide what the analyst considers to be "adequate coverage" of the given domain. By "adequate coverage" I mean that the rules collectively provide sufficient coverage in the domain – that solutions can be generated for sufficient numbers and varieties of problems in the domain.

Armed with the UCM and a sufficient set of higher (and lower) order SLT rules associated with a problem domain, TutorIT will be able to dynamically derive new solution rules as needed. TutorIT will also be able to provide systematic tutoring on all requisite higher as well as lower order SLT rules.

Given any domain, TutorIT's ability to generate solutions will depend on adequacy of requisite Structural Analysis (SA). In this context, it should be emphasized that SA can be applied iteratively. An analyst may build on the results of SA without starting over. SA is a strictly cumulative. The SLT rules deemed sufficient at one point in time may systematically be superseded later on.

Although TutorIT's interface may have to be enhanced, tutoring on higher order SLT rules will take place exactly as any other SLT rule. Knowledge will still be represented hierarchically, and TutorIT decision making will follow the same rules. Critically important from an implementation perspective, theoretical parsimony is matched by the current AuthorIT and TutorIT technologies. It would be fool hardy to underestimate the effort required, but we do not envision major unknowns.

The extended form of TutorIT will select and present problems. The learner will respond, and TutorIT will see if it is correct and provide feedback. If a response is incorrect, TutorIT will provide diagnostic and remediation as detailed above on each of the rules required to solve the problem.

Notice the efficiencies. Suppose the learner is given a complex problem. Instead of having to pinpoint inadequacies in this complex context, it will be sufficient to identify the individual SLT rules and higher order rules necessary for success. Once this has been done, one can treat each individual SLT as before. All SLT rules, higher as well as lower order, have precisely the same formal structure. Hence, diagnosis and remediation can be carried out in modular fashion.

*Comparison with* ITS.-- Given their dominance, comparison with ITS may be helpful to understand the significance of what all this means. AuthorIT can be used to identify what must be learned for success with arbitrary degrees of precision. No longer does one have to worry about individual learner models as such. The author need be concerned only with identifying what higher and lower order SLT rules must be learned for success. This may be done with arbitrary degrees of precision, either initially or in cumulative fashion as experience and development resources dictate. More important perhaps, AuthorIT is not limited in the same way by the complexity of the domain being analyzed. The sheer complexity of some domains

makes them inaccessible to traditional ITS methodology. Traditional ITS development requires coming up de novo with: a) a sufficient set of productions, b) assumptions as to what learning mechanisms to use and c) finally data supporting validity of the analysis.

Structural Analysis does not have the same limitations. What one identifies is whatever an expert in the field believes is necessary and sufficient for success in that domain. Certainly, experts may differ as to what they believe should or might be learned. That is not the point. There is nothing to constrain SA to a single point of view. Complications in supporting multiple perspectives include introducing higher order selection rules for deciding which of the alternative solution rules to use under what conditions. In short, anything that can be done with production systems can be done more simply and with TutorIT in conjunction with higher and lower order SLT rules.

Given a representation of what needs to be learned, whether of just lower order as at present or including higher order knowledge as proposed, TutorIT can quickly and easily construct individual learner models, and maintain them dynamically during the course of tutoring. Most important, an extended TutorIT would be able to address diagnosis and remediation on each SLT rule in strictly modular fashion. The result would be orders of magnitude reduction in (pedagogical) decision making complexity. This is simply not possible in a production systems environment.

As above, TutorIT will work even in the face of incomplete analysis. Even a small amount of analysis is better than little or none. Given its complexity, ITS research can only go so far.

None of this means that we should give up on fundamentals. Most TICL research today is limited to general models or frameworks. Some even come with fancy names. I believe we can do more, however, than introduce acronyms in our research.

We need to concentrate more heavily on identifying what needs to be learned. Fifty years of basic and applied research in the field convinces me that the more precisely one understands what needs to be learned the better job one can do of teaching it. This holds whether one is talking about automated tutorials or human teachers. The only difference is that the former can be automated and are more readily subject to incremental improvement.

*Comparison with Contemporary TICL Research*.-- Compare the above also with what is currently done in contemporary tutoring and simulation systems. In such systems, so-called scaffolding is typically indirect, or at best imprecise. We used to call them “Hints”. Hints can certainly encourage, indeed require involvement of the learner. If successful, they may also exercise the learner’s cognitive abilities.

The problem here is that existing systems of this type have never achieved results comparable to what a skilled human tutor might do. Given increasingly precise representations of what must be learned for success, on the other hand, TutorIT will be capable of providing arbitrarily precise instruction.

Encouraging learners to exercise whatever they may (or may not know) is a good thing. Nonetheless, two points need to be emphasized.

1. There is nothing that forces TutorIT to be as precise as may be possible.
2. There will inevitably learners for which typical scaffolding is insufficient.

AuthorIT’s support for arbitrarily precise representation, extended to include higher order knowledge, will make it possible to reach those who are unable to succeed on their own. By

design, the proposed extension of TutorIT would be capable of precise diagnosis and remediation on higher as well as lower order knowledge.

In this regard, I am tempted to call attention to an early piece of research on math learning (e.g., Roughead & Scandura, 1968) in which we were able to explicitly identify the higher order rules necessary for success in math problems solving (number series). Once identified, we found that students could be taught those higher order rules directly by exposition. Furthermore, it was impossible to tell the difference between those who were taught the higher order rules by exposition and those who discovered them on their own.

I do believe that student's who discover rules on their own and students who are taught those rules secondarily exercise and may learn additional skills. Students who discover higher order rules may also exercise still higher level skills. Conversely, students who learn by exposition gain more experience understanding complex instruction. The essential point is that being able to identify such higher order knowledge (with arbitrary degrees of precision) inevitably makes it easier for more children to learn such higher order skills. Here, we have another example of where computers might eventually take over tasks that they can do better than humans.

*Further Extensions*.—Although discussion is beyond the current scope, it is worth noting that these ideas have implications far beyond tutoring systems. As discussed in Scandura (2007) essentially all expert systems are based on deriving implications from sets of productions governed by learning mechanisms of one sort or another. It would be interesting to compare results of expert systems based on productions + mechanisms versus SLT + UCM. Similarly, it would be nice to compare benefits in automatic problem solving. For that matter, it would be interesting to compare the proposed approach based on KR in SLT and UCM in areas as diverse as robotics and manufacturing

Where do we go from here? Supporting complex domains will not come without a price. Although our current research makes viability clear, the time and effort required with complex domains will almost certainly be greater than with well-defined domains. Mastery in such domains requires the acquisition of higher as well as lower order knowledge. Identifying such knowledge is not always easy. But as early research demonstrates, this can be done (Roughead &Scandura, 1968; Scandura, 1974, 1977; Scandura et al, 1971c; Scandura & Scandura, 1980). Moreover, the process is now far more systematic and it is a task that is long overdue. I leave the position of TICL Chair this year, and am perhaps at the stage of my career where the term "senior advisor" takes on a double meaning.

Having said that, I want to emphasize that what have been talking is not farfetched dreaming. TutorIT technologies have the potential of revolutionizing the way tutorials are developed and delivered. Not only can they be used to develop math skill tutors but highly adaptive tutorials in essentially any area: mathematics, reading, science or otherwise.

We are working hard to bring a critical mass of TutorIT math tutorials to fruition in the very near term. We also have developed a unique viral approach to letting the world know what we have. I invite those of you who may be interested in learning more to meet with me in this room tomorrow afternoon during TICL's off-hours demonstration session at 4 PM.

Together, I believe we can make a real difference. In addition to AuthorIT and TutorIT technologies, we have now for the first time in 50 years a unique means of distribution. Anyone will be able to get free TutorIT tutoring time by going to www.TutorITmath.com. Furthermore, users can earn still more free time by referring others -- who will also get free time. It will be exciting to see the results of half a century of research finally making a difference.

References

Anderson, J. R.: 1993, 'Rules of the mind'. Hillsdale, NJ: Erlbaum.

Anderson, J. R., Boyle, C. F., Corbett, A. T. and M.W. Lewis: 1990, 'Cognitive modeling and intelligent tutoring'. *Artificial Intelligence*, 42, 7-49.

*Bloom, B. S. The 2 Sigma Problem: the Search for Methods of Group Instruction as Effective as On-to-One Tutoring. Educational Researcher, 13, 6, pp.4-16.*

Dijkstra, S., Schott, F., Seel, N., Tennyson, R. D. Mahwah, NJ: Erlbaum, 1997.(Routledge, 1985).

Durnin, J.H. & Scandura, J.M. An algorithmic approach to assessing behavior potential: Comparison with item forms and hierarchical analysis. *Journal of Educational Psychology*, 1973, *65*, 262-272.

Ehrenpreis, W. & Scandura, J.M. Algorithmic approach to curriculum construction: A field test. *Journal of Educational Psychology*, 1974, *66*, 491-498.

Foshay, R. & Preese, F. Do We Need Authoring Systems? A Commercial Perspective. *Technology, Instruction, Cognition & Learning (TICL).* 2005, 2, 3, 249-260.

Foshay, R. & Preese, F. Can We Really Halve Development Time: Reaction to Scandura's Commentary. *Technology, Instruction, Cognition & Learning (TICL).* 2006, 3, 1-2, 191-194.

Gagne, R. M. Conditions of Learning. NY: Holt, Rinehart & Winston, 1965 (First Edition).

Greeno, J. G. & Scandura, J. M. All-or-none transfer based on verbally mediated concepts. *Journal of Mathematical Psychology*, 1966,*3*, 388-411.

Koedinger, K. R., Anderson, J. R., Hadley, W. H. and M.A. Mark: 1997, 'Intelligent Tutoring Goes to School in the Big City'. *Int. J. Artificial Intelligence in Education*, 8, 30-43.

Merrill, M. D. *Principles of Instructional Design*, Educational Technology Publications, 1994, 465 pp.

Mitrovic, A., Koedinger, K. and B. Martin: 2003, 'A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling'. In: P. Brusilovsky, A. Corbett, F. de Rosis (eds) *Proc. 9th Int. Conf. User Modeling*, Springer-Verlag, LNAI 2702, pp. 313-322.

Murray, T. An Overview Of Intelligent Tutoring System Authoring Tools: Updated Analysis of the State of the Art. Chapter 17 in T. Murray, S. Blessing & S. Ainsworth (Eds.), Authoring tools for advanced technology learning environments, The Netherlands: Kluwer, 2003.

Newell, A. & Simon, H.A. Problem Solving. Englewood Cliffs, NJ: Prentice Hall, 1972.

Ohlsson, S. & Mitrovic' A. Fidelity and Efficiency of Knowledge Representations for Intelligent Tutoring Systems. Technology, Instruction, Cognition & Learning (TICL). 2007, 4, 2-4, 101-132.

Paquette, G. Graphical Ontology Modeling Language for Learning Environments. Technology, Instruction, Cognition & Learning. 2007, 5, 2-4, pp. 133-168.

Polya, G. Mathematical discovery (Volume I)., NY: Wiley, 1962.

Reigeluth, C. M. *Instructional Design Theories and Models*. Mahwah, NJ: Erlbaum, 1983.

Reigeluth, C. M. *Instructional Design Theories in Action*. Mahwah, NJ: 1987.

Ritter, S. Authoring model-tracing tutors. *Technology, Instruction, Cognition & Learning*, 2005, 2, 231-247.

Roughead, W.G. & Scandura, J.M. "What is learned" in mathematical discovery. *Jr. Educational Psychology*, 1968, *59*, 283-298.

Scandura, J. M. The teaching-learning process; an exploratory investigation of exposition and discovery modes of problem solving instruction. Dissertation Abstracts, 1963, *23, No. 8*, 2798.

Scandura, J. M. Abstract card tasks for use in problem solving research. *Journal of Experimental Education*, 1964, *33*, 145-148. (a)

Scandura, J. M. An analysis of exposition and discovery modes of problem solving instruction. *Journal of Experimental Education*, 1964, *33*, 149-159. (b)

Scandura, J.M., Woodward, E., & Lee, F. Rule generality and consistency in mathematics learning. *American Educational Research Journal*, 1967, *4*, 303-319.

Scandura, J.M. Learning verbal and symbolic statements of mathematical rules. *Journal of Educational Psychology*, 1967, *58*, 356-364.
Scandura, J. M. & Roughead, W. G. Conceptual organizers in short-term memory. *Journal of Verbal Learning and Verbal Behavior*, 1967, *6*, 679-682.
Scandura, J.M. Concept dominance in short term memory. *Journal of Verbal Learning and Verbal Behavior*, 1967, *6*, 461-469.
Scandura, J.M. & Durnin, J.H. Extra-scope transfer in learning mathematical rules. *Journal of Educational Psychology*, 1968, *59*, 350-354.
Scandura, J.M. The role of rules in behavior: Toward an operational definition of what (rule) is learned. *Psychological Review*, 1970, *77*, 516-533.
Scandura, J.M. The role of higher-order rules in problem solving. *Journal of Experimental Psychology*, 1974, *120*, 984-991.
Scandura, J. M. Deterministic Theorizing in Structural Learning: Three levels of empiricism, Journal of Structural Learning, 1971a
Scandura, J. M. Mathematics: Concrete Behavioral Foundations. NY: Harper & Row, 1971b.
Scandura, J. M., Durnin, J.H. & Ehrenpreis, W. Na Algorithmic approach to Mathematics: Concrete Behavioral Foundations. NY: Harper & Row, 1971c.
Scandura, J. M. Structural Learning 1: theory and Research, London//NY: Gordon & Breach Sci. Pub., 1973.
Scandura, J.M. The role of higher-order rules in problem solving. *Journal of Experimental Psychology*, 1974, *120*, 984-991. (a)
Scandura, J.M., Durnin, J.H., & Wulfeck, W.H., II. Higher-order rule characterization of heuristics for compass and straight-edge constructions in geometry. *Artificial Intelligence*, 1974, *5*, 149-183. (b)
Scandura, J.M. *Problem Solving: a Structural / Process Approach with Instructional Implications.* New York: Academic Press, 1977.
Scandura, J.M. & Scandura, A.B. *Structural Learning and (Piagetian) Concrete Operations*. NY: Praeger Sci. Pub., 1980.
Scandura, J.M. A cognitive approach to software development: The PRODOC environment and associated methodology. In D. Partridge (Ed.): Artificial Intelligence and Software Engineering. Norwood, NJ: Ablex Pub., 1991, Chapter 5, pp. 115-138.
Scandura, J.M. Automating renewal and conversion of legacy code into ada: A cognitive approach. *IEEE Computer,* 1994, April, 55-61.
Scandura, J.M. Cognitive analysis, design and programming: generalization of the object oriented paradigm. Valley Forge, PA: *Proceedings of the National Ada Conference,* March 1995.
Scandura, J.M. and Scandura, Alice B. Improving RAM in Large Software System Development and Maintenance. *Journal of Structural Learning and Intelligent Systems*, 1999, *13*, 227-294.
Scandura, J.M. Structural (Cognitive Task) Analysis: An Integrated Approach to Software Design and Programming. *Journal of Structural Learning and Intelligent Systems* (a special monograph), 2001, 14, 4, 433-458.
Scandura, J. M. AuthorIT: Breakthrough in Authoring Adaptive and Configurable Tutoring Systems? *Technology, Instruction, Cognition & Learning (TICL),,* 2005, 2, 3, 185-230.
Scandura, J.M. How to Cut Development Costs in Half: Comment on Foshay &* Preese. *Technology, Instruction, Cognition & Learning (TICL),* 2006a, 3, 1-2, 2006, 185-190.
Scandura, J.M. Reaction to Foshay & Preese Reaction. *Technology, Instruction, Cognition & Learning (TICL),* 2006, 3, 1-2, 2006b, 195-197.
Scandura, J. M. AST Infrastructure in Problem Solving Research. *Technology, Instruction, Cognition & Learning (TICL),* 2006c, 3, 3-4, 1-13.
Scandura, J. M. Learning Objects: Promise versus Reality. *Technology, Instruction, Cognition & Learning (TICL),* 2006d, 3, 3-4, 25-31.

Scandura, J. M. Converting Conceptualizations into Executables: Commentary on Web-based Adaptive Education and Collaborative Problem Solving. *Technology, Instruction, Cognition & Learning (TICL).* 2006e, 3, 3-4, 345-354.

Scandura, J. M. Knowledge Representation in Structural Learning Theory and Relationships to Adaptive Learning and Tutoring Systems. *Technology, Instruction, Cognition & Learning (TICL)*, 2007, Vol. 5, pp. 169-271.

Scandura, J. M. Introduction to Knowledge Representation, Construction Methods, Associated Theories and Implications for Advanced Tutoring/Learning Systems. *Technology, Instruction, Cognition & Learning (TICL)*, 2007, Vol. 5, pp. 91-97.

Scandura, J. M., Koedinger, K, Mitrovic, T, Ohlsson, S. & Paquette, G. Knowledge Representation, Associated Theories and Implications for instructional Systems: Dialog on Deep Infrastructures, *Technology, Instruction, Cognition & Learning (TICL)*, *2009, 6.* pp.125-149*.*

Software Engineering Publications

Scandura, J.M. Cognitive technology and the PRODOC re/NuSys Workbench™: a technical overview. *Journal of Structural Learning and Intelligent Systems*, 1992, *11*, pp. 89-126.

Scandura, J.M. A cognitive approach to software development: The PRODOC environment and associated methodology. In D. Partridge (Ed.): Artificial Intelligence and Software Engineering. Norwood, NJ: Ablex Pub., 1991, Chapter 5, pp. 115-138.

Scandura, J.M. Automating renewal and conversion of legacy code. *Software Engineering Strategies*. NY: Auerbach Publications, 1994, March/April, pp. 31-43. (Reprinted in *Handbook of Systems Management, Development and Support*, Auerbach, 1995. Similar version in Scuola estiva: Engineering of existing software. Bari, Italy: Dipartimento di Informatica Universita degli Studi di Bari, 1994, pp. 179-192*.)*

Scandura, J.M. Automating renewal and conversion of legacy code into ada: A cognitive approach. *IEEE Computer,* 1994, April, 55-61.

Scandura, J.M. Cognitive analysis, design and programming: next generation OO paradigm. *Journal of Structural Learning and Intelligent Systems,* 1997, 13, *1*, 25-52.

Scandura, J.M. A cognitive approach to reengineering. *Crosstalk, The Journal of Defense Software Engineering*, June 1997, 10, *6*, 26-31.

Scandura, J.M. and Scandura, Alice B. Improving RAM in Large Software System Development and Maintenance. *Journal of Structural Learning and Intelligent Systems*, 1999, *13*, 227-294.

Scandura, J.M. Structural (Cognitive Task) Analysis: An Integrated Approach to Software Design and Programming. *Journal of Structural Learning and Intelligent Systems* (a special monograph), 2001, 14, 4, 433-458*.*

*Key Patent & Pending*

Scandura, J.M. U.S. Patent No. 6,275,976. *Automated Methods for Building and Maintaining Software Based on Intuitive (Cognitive) and Efficient Methods for Verifying that Systems are Internally Consistent and Correct Relative to their Specifications*. August 14, 2001.

Scandura, J.M. Method for Building Highly Adaptive Instruction. US Patent pending.

## Appendix A

The basic steps in SA (detailed in Scandura, 2007, in press) are carried out by subject matter experts (SME) as follows:

1. Start with an Informally Defined Problem Domain: Select a Representative Sample of Prototypic Problems in Domain
2. Systematically Construct SLT Rules for Solving Prototypic Problems
3. Convert SLT Rules into Higher Order Problems
4. Construct Higher Order SLT Rules for Solving these Higher Order Problems
5. Optionally Eliminate Redundant SLT Rules
6. Repeat Process Until Desired Level of Domain Coverage Is Attained

Complex domains include problems that cannot be solved by any individual SLT rule that has been taught or learned. Currently, AutoBuilder only supports that part of SA needed in the analysis of well-defined domains -- steps 1 and 2 where all problems in the domain can be solved by discrete SLT rules. The screenshot Fig. 1 illustrates this with simple column subtraction.

In general, steps 1 and 2 are carried out by the SME using AutoBuilder. In step 1 in the SME selects prototypic problems in some informally defined domain. In step 2 the SME systematically constructs SLT rules for solving the selected problems (e.g., column subtraction, adding fractions, solving quadratic equations, etc.).

Structural Analysis (SA) of complex problem domains has been used successfully by SMEs to identify higher order (e.g., heuristics) SLT rules for some time (Scandura at al, 1971, 1974, 1977). Until recently (cf. Scandura, 2001, 2007), however, Step 3 has remained largely intuitive with little chance of automated assistance. Consequently, AutoBuilder cannot currently deal with the underlined step 3. Demonstrating the feasibility of extending AutoBuilder to support the systematic construction of higher order SLT rules will be the primary task during Phase I.

Step 3 presents the primary challenge. Step 3 in SA involves converting SLT rules into higher order problems is critical in analyzing complex domains. This step requires: (a) replacing specific nodes in a SLT solution rule with abstractions and (b) selecting simpler available rules from which the SLT solution rule can be constructed. Once Step 3 has been completed Step 4 is identical to Step 2.

Steps 1-6 are illustrated in the following three examples, with emphasis on steps 3 & 4 – which are essential in dealing with higher order knowledge. These examples illustrate how higher order SLT rules are derived using three simple domains: Measure Conversion, Proofs in Trigonometry and Number Series. Only top level SLT rules are shown below: Once a top level SLT rule has been constructed, refinement proceeds as above with all SLT rules, whether higher or lower order.

**Example 1: Measure Conversion**

**1.SME Selects Prototypic Problems**

***3 yd*** **--** ***?in***

***2 gallon*****--** ***?pints***

**2. Construct Solution Rules for Prototypic Problems**

***yd*** ***→36_times→*** ***in***

***gallons*** ***→8_times→*** ***pints***

**3. Convert SLT (solution) Rule to Higher Order Problem**
(Construct Goal & Given of Higher Order Problem)

Givens: *yd* →*n_1_times*→ *xxx*
*xxx* →*n_2_times*→ *in*

Goal: *blug* →*n_times*→ *clug*

**4. Construct SLT Higher Order Rule Composition Problem**

**(Domain/Range Structure of H. O. Rule is Un-initialized Version of Higher Order Problem)**

DOMAIN*: *blug [n_times] xxx*
*xxx [n_times]) clug*

RANGE: *blug (n_times) clug*

**Construct Procedure for Higher Order SLT (Composition) Rule**

PROCEDURE: *compose rules so output of first matches input to second*

**Example 2: Proving Trigonometry Identities**

**1. SME Selects Prototypic Problem (one of many)**

$\sin^2 A + \cos^2 A = 1$ -- ?proof

**2. Construct Solution Rules for Prototypic Problems**

$\sin^2 A + \cos^2 A = 1$ → divide $a^2 + b^2 = c^2$ by c, substitute sin, cos definitions→ Proof is resulting steps

**3. Convert SLT Rule to Higher Order Problem (Replace Semantic-specific Nodes in Solution Rule with Abstractions & Select Rule(s)**

Given

Goal

$\sin^2 A + \cos^2 A = 1$ → divide $a^2 + b^2 = c^2$ by c, substitute sin, cos definitions→ Proof is resulting steps

Trig Identity → divide $a^2 + b^2 = c^2$ by side, substitute trig. fn. Definitions→ Proof is resulting steps

**4. Construct H.O. SLT Generalization Rule**
→Replace Specific Values (e.g., c, sin) with Generalizations→
(e.g., c→side; sin→trig_fns)

**Example 3: Number Series**

**1. SME Selects Prototypic Problems (one of potentially many)**

*1 + 3 + 5* -- *?sum*

**2. Construct (multiple) Solution Rules (2A, 2B, 2C) for Prototypic Problem**

*2A* *1 + 3 + 5* →*3x3*→ *9*
*2B* *1 + 3 + 5* →*3x(1+5)/2*→ *9*
*2C* *1 + 3 + 5* →*successive addition*→ *9*

**3. Convert SLT Rule to Higher Order Problem**
(Construct Goal & Given of Higher Order Problem)

**Higher Order Problem 3A:**

*1 + 3 + 5* →*3x3*→ *9*
*1 + 3 + 5 + 7 + … →nxn→* *Sum*

**Higher Order Problem 3B:**

*1 + 3 + 5* →*3(1+5)/2*→ *9*
*a + a+d + … + L=a+(n-1)d* →*n(a+L)/2*→ *Sum*

**Higher Order Problem 3C:**

*1 + 3 + 5* →*1+3+5*→ *9*
$a_1 + a_2 + a_3 + \ldots + a_n$ →*successive addition*→ *Sum*

**4. Alternative SLT Higher Order Rules for Solving Higher Order Problems 3A, 3B, 3C**

**Higher Order Rule 3A:** →replace 3 terms by n→
**Higher Order Rule 3B:** →replace 1 by a, 5 by l, 3 terms by n→
**Higher Order Rule 3C:** →replace each term by a variable, three terms by n→

Notice that each higher order SLT rule has a different domain of applicability. Higher order rule 3A is very efficient but only works with arithmetic series beginning with 1 and having a common difference of 2 -- for example, *1 + 3 + 5 + … + 99 →50x50→ 2500.* Rule 3B is reasonably efficient and works with all arithmetic series. Rule 3C is relatively inefficient (especially with long series) but works with all number series, arithmetic or otherwise. *(NOTE: For early empirical research on the subject see Scandura, Woodward & Lee 1967; Scandura 1967.)*

In effect, there are three applicable rules in Example 3. Deciding which one to use is essentially what one does in many design problems, where there are alternate ways of solving a problem and selecting which one to use plays a critical role. The process of selection represents a still higher order problem. The given in the problem consists of the three alternative rules. The goal is to select exactly 1. One higher order SLT selection rule that works can be summarized as:

**Case Type-of- Number Series:**

**a) starts with 1 with a common difference of 2 → select rule $N^2$**
**b) common difference → select rule N(A+L)/2**
**c) else →select successive addition**

A more general but error-prone selection rule is to simply choose the simplest rule. Domain of applicability was largely ignored in early research. Defining the domain structures associated with higher order SLT rules plays an essential role, however, in determining which rules to use under what circumstances.

The process of SA can be repeated (indefinitely). Step 5 makes it possible (optionally) to eliminate redundant SLT rules -- e.g., rules like 4x4, 5x5, …, 50x50 can be derived by applying higher order rule 3A, for example, to 3x3. Higher order rules make it possible to derive any number of new SLT rules from basic rules. A wide variety of conversion problems, for example, can be solved by combining a small number of basic volume, weight, currency, etc. equivalents. Repeating the process (step 6 in SA) increases the generative power of the SLT rules and higher order rules associated with the ill-defined domain. Analysis of several complex domains (e.g., Scandura et al, 1974, Scandura, 1977, Scandura & Scandura, 1980) shows that as SA proceeds two things happen: The individual rules become simpler but the generating power of the rule set as a whole goes up dramatically, thereby expanding coverage in original domain (esp. see Scandura, et al, 1977; Wulfeck & Scandura, 1977).

To summarize, the Measure Conversion domain in Example 1 includes any number of (known & unknown) conversion problems, all solvable by **chaining** one known role after another. Example 2 outlines a method (higher order SLT rule) for deriving trigonometric identities as **generalizations** of the Pythagorean theorem (similar to Case Based Reasoning). Example 3 illustrates an ill-defined domain where alternative SLT solution rules are commonly taught (or otherwise learned). As above, this leads to identification of higher order SLT selection rules. It is exactly these kinds of selection rules that must be learned to make sound decisions, whether it be in solving verbal problems in mathematics, or otherwise**.**