What TutorIT Can Do Better Than a Human and Why: Now and in the Future*

JOSEPH M. SCANDURA

Director of Research, MERGE Research Institute and
Emeritus Professor, University of Pennsylvania

More and more things that humans used to do can be automated on computer. In each case, complex tasks have been automated – not to the extent that they can be done as well as humans, but better.

I will draw and develop parallels to education – showing how and why advances in the Structural Learning Theory (SLT) and the AuthorIT development and TutorIT delivery technologies based thereon make it possible not only to duplicate many of the things that human tutors can do but to do them better. Specifically, I will show how and why TutorIT can now do a better job than most if not all human tutors in providing more effective and efficient tutoring on essentially any well defined skill. I also will show why this approach has the potential to also match or exceed human tutoring on ill-defined learning in the future.

Keywords: Structural Learning Theory, AuthorIT, TutorIT, Intelligent Tutoring Systems, ITS, automated tutoring, adaptive tutoring systems, structural analysis, SLT rules, AutoBuilder, Softbuilder, configurable tutoring systems.

Automation involves the use of control and information systems to reduce the need for human intervention. According to Wikipedia, automation is a step beyond mechanization. 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. AI, for example, was founded on the claim that a central property of human intelligence

* This is the full text of an invited address, which by its nature raises fundamental questions about the role of automation in instruction, and presents a way forward. Short term solutions are detailed, and longer term extensions are outlined – all with fundamental implications for the role of automation in instruction.
can be so precisely described that it can be simulated by a machine.\(^1\) 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, where 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? … Today, 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. In each case, *increasingly complex tasks have been automated* – not to the extent that they can be done as well as humans, but better.

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

Other tools such as Texas Instrument’s TI-Nspire tackle the problem from the opposite direction. Rather than hints, calculators serve as tools students can use to 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. A major problem is that good tutoring systems have been difficult and expensive to build. Moreover, their educational benefits have been 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, and comparisons with classroom instruction are often hard to evaluate.

---

\(^1\) This definition derives from John McCarthy’s view of AI “the science and technology of making intelligent systems”. Early researchers at Carnegie Mellon (Newell & Simon, 1972) tended to view AI more in terms of simulating human thought – as trying to describe human cognition in precise terms similar to those required to program a computer, believing that doing so would help to reveal fundamental properties of human intelligence.
Contemporary instructional systems deriving from instructional design principles help (cf. Reigeluth, 1983; Merrill, 1994; Dykstra et al, 2001). Among other things instructional design models 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.

Task analysis, for example, helps identify the order in which prerequisites must be learned. Categories of learning (cf. Gagne, 1965) focus on conditions necessary for specific kinds of learning (e.g., facts, concepts and rules). Both leave major gaps. Many kinds of learning involve various kinds of learning. Relational models (e.g., Paquette, 2007, 2009; Seel, 2003) help fill the gap. The number of relationships, however, quickly expands with complexity (e.g., Scandura, 2005, 2007).

Perhaps the single most important limitation of instructional design models is the large gap between knowledge representation and instructional systems based on those representations. Indeed, the gap is so large that one can start with fundamentally different theoretical models and end up with instructional systems that behave in essentially the same way (e.g., Seel, 2003).

After years of effort, beginning with Control Data’s work (under the leadership of William Norris) in the early 1960s, the best CBI (e.g., ALEKS, a McGraw-Hill Company) is still 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.

Intelligent Tutoring Systems (ITS) systems go further. ITS attempt to duplicate or model what human tutors can do – by adjusting diagnosis and remediation dynamically during instruction. They 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 elements may be available in learner minds (e.g., productions or constraints) and learning mechanisms controlling the way those elements 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 (i.e., more closely aligned with what goes on in classrooms).

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. It is not surprising in this context that Ohlsson and Mitrovic found common cause in developing Constraint Based Modeling (CBM,
CBM essentially is a simpler alternative to ITS based on production systems. In CBM the focus is on constraints that must be met during the course of instruction – not on cognitive constructs (productions) responsible (for meeting those constraints). See Mitrovic & Ohlsson (2007) and Scandura, Koedinger, Mitrovic, Ohlsson, & Paquette (2009) for detailed discussion of the relative merits of each position.

The bottom line: From 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 Structural Learning Theory (e.g., Scandura, 2007, Scandura et al, 2009, hereafter SLT) and AuthorIT and TutorIT technologies (Scandura, 2005) 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. Using “Flash cards” to memorize simple facts, for example, could easily be replaced by computers – with more efficiency and certain results.

This paper goes further. I will show:

a) That AuthorIT now makes it possible to create and that TutorIT now makes it possible to deliver highly adaptive (and configurable) tutoring on basic math skills, indeed on essentially any well-defined skill, all automatically.
b) Why and in what sense TutorIT tutorials can guarantee mastery of such skills, and why they can do this as efficiently as any human tutor.
c) Why TutorIT tutorials can be developed cost effectively – at half the cost of traditional CBI development, and at a fraction of the cost of other ITS systems.
d) How TutorIT tutorials can gradually be extended to support the development and delivery of higher as well as lower order knowledge.

The paper is organized as follows:

1. **Background:** A short summary of thinking that led to Structural Learning Theory (SLT) and how and why SLT has evolved over the years.
2. **Theoretical Advances: Well-Defined Knowledge:** A summary of recent advances in knowledge representation and SLT offering a theoretically rigorous, empirically sound foundation for building highly adaptive tutoring
What TutorIT Can Do Better Than a Human and Why: Now and in the Future

systems, and why these advances make it possible to exceed human tutoring on well defined knowledge.

3. Current Status of AuthorIT & TutorIT – Guaranteed Learning and Lower Cost: How AuthorIT makes it possible to develop highly effective TutorIT tutorials at greatly reduced cost – even less than commercial development. This section also explains how TutorIT math tutorials work, why they can be expected to do as good if not better job than most human tutors, and how they can easily be configured to meet alternative needs (e.g., to serve as diagnostic as well as instructional systems and/or to reliably compare alternative pedagogies).

4. Critical Advances in Current SLT Theory: A summary of recent advances in SLT making it possible to support tutoring on higher order knowledge, including ill-defined learning.

5. AuthorIT and TutorIT Extensions: How AuthorIT and TutorIT can be extended to support adaptive tutoring on complex and/or ill-defined content domains where higher order knowledge plays a central role?

1 BACKGROUND

In the 1960s, there was a disconnect in educational research and research in subject matter (math) education. Educational research was experimental in nature and focused on behavioral variables: exposition vs. discovery, example vs. didactic, demonstration vs. discussion, text vs. pictures, look-say vs. phonics, 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, what little research there was in subject matter (e.g., math) education, focused primarily on what content should be taught (e.g., traditional versus “new math”).

During the same time period, the emerging field of instructional design focused on various kinds of learning and prerequisites for same. Task analysis focused initially on learner behavior (Miller, 1959; Gagne, 1966). My original work in cognitive task analysis (e.g., Greeno & Scandura, 1966; Scandura & Roughead, 1967, Scandura, 1970, 1971, Durnin & Scandura, 1973) added the critical dimension of what needed to be learned to produce desired behavior.

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 et al, 1974; Scandura, 1977).

My research from inception to current has 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, informally defined → **Now**: Universal Control Mechanism (UCM), formally defined]

- **Assessing Behavior**: How can one determine what an individual does and does not know?
  
  [1970: Which paths in directed graphs are known → **Now**: which nodes in AST are known (+), unknown (−), to be determined (?)]

- **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]

Both higher order and lower order knowledge played a central role in SLT 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 needs 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/or 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 & Roughhead, 1967; Scandura, 1967), I found unsatisfying comparisons based on averaging behavior over multiple subjects. I wanted something better – more akin to what had been accomplished in
physics centuries earlier (cf. Scandura, 1971, 1974a). SLT was unique when introduced, and raised considerable interest both in America and internationally (Scandura, 1971a, 1973, 1977).

The mid-1970s, however, also saw emergence of the fledging field of cognitive psychology, then a relatively direct extension of experimental psychology with an AI influence. Cognitive psychology also discovered the importance of content. Instead of operationally defining knowledge in terms of behavior, however, cognitive psychology often equated theory with alternative ways of representing knowledge. Research focused on making assumptions about what (productions or relationships) might be in learner minds and comparing fit with observable behavior. Experimental studies invariably followed the traditional statistical paradigm.

In the later 1970s and early 1980s, we developed hundreds of Computer Based Instruction (CBI) programs inspired but not based on SLT. Many sold for decades making small publisher Queue one of Inc Magazine’s 100 fastest growing small businesses. Nonetheless, subsequent development in CBI became increasingly 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 included but 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 independent study, the long and short of it is that, SLT failed to significantly inform on-going research in either CBI or ITS. After the interdisciplinary doctoral program in structural learning at Penn was eliminated in the early-mid 1970s, SLT became a poorly understood historical curiosity for several decades.

Recent publications in TICL have led to an increased understanding of SLT and its relationships to ITS, as well as to CBI based on relational and hierarchical models generally (e.g., Mitrovic & Ohlsson, 2007; Paquette, 2007; Scandura, 2007; cf. respective advantages and limitations in Scandura, Koedinger, Mitrovic & Ohlsson, Paquette, 2009). The long and short of it is that recent advances in the

---

2 The deterministic philosophy proposed in SLT represents a major departure in thinking about how to evaluate learning and behavior. Just as classical physics ignored the effects of friction in its laws of falling objects, for example, SLT focused on behavior under idealized (e.g., memory free) conditions. After understanding how TutorIT works, please see my concluding comments on this subject.
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\(^3\) 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 the 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 widely used in our PRODOC and Flexsys software development systems (e.g., Scandura, 1991, 1994). To date, ASTs have had almost no impact on knowledge representation in either ITS or CBI. We will soon see, however, that ASTs do have very significant advantages.

I have recently documented the current form of SLT in some detail (Scandura, 2007). Readers are encouraged to review the material therein along with 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 this representation offers major advantages in developing adaptive tutoring systems.

2 THEORETICAL ADVANCES: WELL-DEFINED KNOWLEDGE

There have been three fundamental advances in SLT in recent years. First is 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 complex ill-defined domains.

\(^3\) I used the term “rule” rather extensively in behavioral research during the 1960s. Adopting the term “production” from the logician Post in the 1930s, Newell & Simon (1972) introduced the term “production rule” in their influential book on problem solving. Anderson later used the term “rule” in ITS as synonymous with “production rule” (in production systems). To distinguish the two I ultimately introduced the term “SLT rule”. Distinctive characteristics of SLT rules became even more important with my introduction of SLT in SLT. In this context, ASTs represent a long sought solution to my early attempts at formalization in SLT (see Scandura, 1973, Chapter 3). The importance of ASTs in SLT, however, only gradually became clear to me after using the concept for some time in developing our software engineering tools – despite the fact that ASTs had played a central role in compiler theory for years.
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 generate both test problems and the solutions to those test problems. As we shall see, they also make it possible to simultaneously represent knowledge at multiple levels of abstraction.

**Precision**

A 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 of both traditional CBI (cf. Paquette, 2007) and ITS (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.

There is no need 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 on well-defined tasks.

Specifically, Structural (domain) Analysis (e.g., Scandura, 2007) makes it possible not only to represent all behavior hierarchically, but to do so with arbitrary degrees of precision ensuring contact with assumed prerequisites.
It is well 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 subsets, or 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 the set of “houses”. Each house in the set consists of room elements. Rooms, in turn, can be categorized by their size, or their use, or by any number of other distinctions.

Component and category refinements each have direct counterparts in corresponding solution procedures (Scandura, 2001, US Patent 6,275,976). Again, consider column subtraction. The initial procedural refinement is a Repeat-Until loop. Loops in procedures correspond precisely to Prototype refinements in data (in the problems on which they operate): 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” similarly 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 have a 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, the number of relationships on relationships can extend geometrically without bound.
Relational representations suffer from an additional problem (beyond the sheer number of relationships). We shall see that component and category refinements may be repeated in the same way indefinitely. This is not possible with (non-unary) relationships. Every relationship (relational refinement) must be considered anew. There is no uniform way to represent given (non-unary) relationships in terms of simpler elements.

Knowledge representation using ASTs offers a way to solve 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 (SA), 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” (which consists of relationships between line segments).

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 strictly 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 well defined domain in whatever detail may be necessary (and/or desirable).

**NOTE 1:** 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 (i.e., variables) in a relationship.

**NOTE 2:** It might appear that arbitrary refinement may be as, if not more demanding than knowledge engineering in ITS. Identifying possible (correct and/or error) productions, however, not to mention learning mechanisms, can be very challenging and open ended. On the other hand, the process of Structural Analysis (SA), is highly systematic with a definitive end point. In addition to learning how to perform SA using AuthorIT’s AutoBuilder
component (see below), the main requirements for an author are the ability to perform the skill in question and reasonable insight into what must be learned for success. Working under my direction a single programmer familiar with AuthorIT and TutorIT has been making TutorIT tutorials ready for field testing at a rate of at least one per month. Only a small portion of this time has involved representing to-be-acquired knowledge. Most has been devoted to and/or technical requirements in making that knowledge executable, laying out interfaces and associated media. Although explanation is beyond the scope of this article, a very recent breakthrough makes it possible to make Flexforms executable automatically, completely eliminating the need for traditional programming, and making it possible for non-programmers to create TutorIT tutorials in any subject matter.

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 decision making. 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 prerequisite knowledge available to students in 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 the procedure in 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 knowledge associated with any given subtree in an SLT rule.

It is not always feasible, however, nor necessary to undertake complete analysis. 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. This fact has important implications not only for well defined domains but for any complex domain where it is difficult to identify all prerequisites with the precision necessary for complete analysis.

NOTE: Curriculum standards specifying prerequisites, concepts to be learned and the order in which they should be acquired may serve as a starting point. Generally speaking, however, our experience is that they do
not normally go nearly far enough in identifying what must be learned for success. Nonetheless, partial analysis offers a way forward in addressing concerns raised by DuBoulay (this issue). I trust this important issue will be discussed at the forthcoming TICL symposium.

It is 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 to any desired degree of certainty.

### Efficiency of Development

A major limitation of adaptive tutoring systems is that they have been hard to build. Identifying knowledge is only one part of the process. Defining (and implementing) pedagogical decisions – what to test or teach and when – is the single most expensive, time consuming and error prone task in traditional ITS development (Mitrovic & Ohlsson, 2007; Koedinger & Ohlsson, 2009). This perhaps is the primary reason so few truly adaptive tutoring systems exist despite many years of university and 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. This simple fact has made it possible to develop a critical mass of TutorIT tutorials in less than a year. A small team of programmers working under my direction has been able to learn to use AuthorIT, to refine AuthorIT and to develop fully adaptive TutorIT tutorials covering a full set of pre-algebra and elementary algebra skills, including basic facts, whole number algorithms, fractions, signed numbers and complex expressions (including use of parentheses). See www.TutorITmath.com for up-to-date particulars.

Cost savings in development in comparison with traditional CBI have been estimated at between 40 and 60% (cf. Foshay & Preese, 2005, 2006; Scandura, 2006a, b). Results to date and recent technical advances suggest the possibility of even greater efficiencies.

Hierarchical representation has a further not inconsequential benefit. It is possible 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).

In short, guaranteed results and multiple delivery modes at lower cost – a combination that should be hard to resist.
3 CURRENT STATUS OF AUTHORIT & TUTORIT – GUARANTEED LEARNING AND LOWER COST

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

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. Terminal nodes correspond to prerequisites, each of which is executable. (NOTE: AST-based Flexforms formally represent a dynamic generalization of XML).
   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 Tool, a dialog 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 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.

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” (Operation [arrow] A) in Flexform 3 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” (Operation B) provide increasingly more specific information.

Parameters of these operations represent data on which these operations act. This data also is arranged hierarchically. Operation A, for example, operates on “Prob” and “CurrentColumn”. “Prob” represents an entire subtraction problem. “Current-Column” represents columns in such problems. Operation B also includes “Reduced-Top”, “Slashtop”, “CurrentBorrowColumn” and “BorrowedDigit” as shown in Figure 2. These latter data nodes are components common to all columns.

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 (with a to-be-computed difference) and generates the current column with the computed 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 procedural nodes operate on (as or) more highly structured parameters. For example, CurrentColumn represents entire columns, including the column itself and the top, bottom, difference and borrow digits in that column. Corresponding lower level child nodes operate on simpler parameters, like BorrowedDigit. The behavior of higher and lower level operations (i.e., nodes), however, is expected to be equivalent (to produce equivalent results). In effect, each level of analysis represents equivalent behavior. The only difference
FIGURE 1
Successive levels of procedural refinement in Column Subtraction. Above are three Flexform screen shots showing increasing 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.

is that higher level procedural nodes operate on more complex structures (defined by higher level data nodes).

In effect, Flexforms represent knowledge at multiple levels of abstraction. Higher level procedure nodes operate on higher level structures and represent knowledge that is more declarative in nature. Lower level nodes operate on simpler data structures and is correspondingly more procedural. All knowledge is inherently both procedural and declarative (structural) in nature. It is simply a matter of degree.

Terminal nodes in a Flexform represent the most elementary levels of analysis. They correspond to the simplest operations and decisions that are assumed to be
known in advance. They correspond to prerequisites assumed to be necessary to master the knowledge represented by the Flexform. In the absence of further analysis, it is assumed that each terminal is either known in advance or can be learned directly via instruction on the given terminal.

Each of these terminal nodes must be made executable on a computer. Execution in the preferred embodiment is enabled by TutorIT’s built-in interpreter. Making knowledge Flexforms executable enables TutorIT to generate solutions to given problems. TutorIT, in turn, uses these solutions to evaluate learner responses. The green nodes below the highlighted (blue) terminal node in Figure 3 are illustrative. The executable nodes are implemented in AuthorIT’s High Level Design (HLD) language.

**Defining Problems**

Once a Flexform has been fully implemented (and tested using AuthorIT’s build in Interpreter/Visual Debugger), the next major step is to define problems 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. 5.

**TutorIT Options Tool**

The TutorIT Options Tool, implemented as a dialog in AuthorIT, is used by authors to define 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 6 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. Among other things, an author can assign desired mastery levels, whether or not to provide feedback, certainty of mastery (requiring multiple successes on individual nodes), the expected initial status of learners on individual nodes, assumed innate abilities to more quickly master more advanced materials (e.g., don’t require mastery of prerequisites before providing instruction) and the order in which nodes are selected for testing or instruction. In short, TutorIT makes it possible to compare different pedagogies on even terms.

**TutorIT**

TutorIT receives the output of AuthorIT as input. 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 Flexform defining the skill acts as a structured database, including all information needed by TutorIT to deliver content in any given delivery mode. In addition to to-be-learned operations and decisions, Flexforms include: a) a modular executable implementation of each terminal (Fig. 3), b) questions, instruction, feedback and corrective feedback associated with specific nodes in the Flexform (Fig. 4), c) problem schemas (which serve as input to the Flexform) laying out the kinds of problems to be solved (Fig. 5), and d) TutorIT options specifying how TutorIT is to make its decisions (Fig. 6)) problem schemas. Users (e.g., teachers or tutors) also are allowed discretion in choosing how tutoring is to be delivered.

TutorIT selects problem schemas one by one as specified. It selects nodes in the given Flexform that will reveal as much as possible about what the learner knows at each point in time. TutorIT generates a sub-problem (of the problem schema) by executing the Flexform up to the selected node. This generates the input to the sub-problem as well as problem nodes that are not yet determined. The latter become outputs or goal nodes for the student.
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. It also shows where instruction, questions, positive and corrective feedback are to appear (some appear in the same position, but not at the same time). 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).

Execution continues through the sub-tree defined by the selected node. This execution thereby generates the solution to the subproblem defined by the selected node. The generated sub-problem is displayed on TutorIT’s Blackboard as specified by the values and attributes assigned to nodes in the problem in the Blackboard Editor during authoring. In Figure 7B the “problem” is presented as a Question. In Figs. 7B and 7C it is presented as instruction. In Fig. 7C, instruction is accompanied by a supporting Flash movie.

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.
FIGURE 6
TutorIT Options Tool used by authors to define/configure alternative learning modes. Currently set to ADAPTIVE mode. Other choices allow for further customization.
FIGURE 7A
Shows TutorIT running in Tutor Mode. In Tutor Mode a Learner Model is displayed along with a number of options giving a teacher or tutor considerable latitude in defining TutorIT’s decision making process. For a student population that has already begun Column Subtraction, the initial status on each node may be set to “?” because each student’s status is yet to be determined.

FIGURE 7B
Shows a sample question with TutorIT run in Student Mode. Notice that there are multiple answer boxes, which the student is yet to complete.
FIGURE 7C
Shows corrective feedback which the student receives after making a mistake. Sound and an accompanying Flash file explain and dynamically illustrate what needs to be done.

Learner Model

TutorIT takes Flexform files as input and creates a Learner Model (see Figure 7A) representing what the learner initially knows or is assumed to know about the to-be-learned Skill. The Learner Model is displayed only when TutorIT is run in “Tutor Mode”. In Tutor Mode, the exact status of the student at each point in time is shown in a tree view with each leaf marked with a “+”, “−” or “?”.

Nodes in the treeview correspond one to one to (blue or actionable) nodes in the Flexform (e.g., see Fig. 1). When run in Tutor Mode, tutors also have an option to fine tune TutorIT’s decision making.

All TutorIT 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. 6).
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 Fig. 7).

5. If a node is marked “−”, TutorIT may be configured to provide instruction. If marked “?”, TutorIT may present a question to determine its status. Alternatively, run in default mode, TutorIT always gives the student a choice. He or she may answer the question (if the answer is known) or ask for help (instruction). TutorIT skips nodes marked “+” unless the node is an automation node. Automation nodes require a faster response (higher level of expertise).

*(NOTE: 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 also must meet timing requirements if the node is an automation node requiring a higher level of mastery.)*

7. In addition to determining the learner’s status on individual nodes, and most importantly, TutorIT also infers what the learner 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. In effect, TutorIT acts as might an skilled tutor with comparable understanding of the skill in question.

TutorIT can be configured with various “safety factors” to ensure learning. For example, one can set the Options Tool to require learners to demonstrate mastery on every node, not just once but any specified number of times (see “Learning [No. successes per node]”). After learning, TutorIT can be set to require any specified level of success on practice problems.

To date, we have developed TutorIT tutorials for the Basic Facts (five levels for each of: Addition, Subtraction, Multiplication and Division), Whole Number Arithmetic (Column Addition, Column Subtraction, Column Multiplication and Long and Short Division), Fractions, Decimals, Signed Numbers, Complex Expressions and Basic Mathematical Processes (pre-algebra skills involving detecting regularities, constructing examples, interpreting mathematical descriptions, describing ideas in mathematics, axiomatization and deduction). All are ready for field testing. TutorIT tutorials for evaluating and simplifying algebraic expressions, and for solving quadratic equations also are nearing completion as this article goes to press. 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 it is 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. Pedagogical decision making that is completely independent of content semantics 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 with respect 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 that 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.

TutorIT tutorials can be unusually effective because they benefit from careful pre-analysis. Moreover, they can be improved incrementally. Unlike busy teachers, with limited time, we have put a significant amount of time into designing each TutorIT Math skill tutorial – far more than what goes into writing a text book for example. They can only get better with feedback from the field.

On the other hand, a good human tutor generally will be more attuned to motivational factors. While TutorIT tutorials may be expected to get better over time as a result of feedback, they are designed for specific purposes. They may never achieve the flexibility of a good human tutor who has spent years both learning math and how to motivate children to learn in a wide variety of real world situations. In short, there likely will always be some things that a good human can do better than TutorIT. But the converse is also true. The choice is not one of either or. Rather it is a question of how to use both in an optimal fashion to maximize learning. (In this context, please see sections 4 and 5 which address both complex problem solving and motivational needs.)

Importance of Prerequisites

One might argue that just because a student solves 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 be sufficient only when analysis is complete – when all terminal nodes are as we say “atomic”. Research shows that success on one subproblem associated with an atomic node implies success on all instances – with unusually high degrees of reliability (cf. Scandura, 1971a, 1973, 1977). Having said this, an author can add redundancy, just as a good bridge designer builds in a safety factor. TutorIT can be required to demand a higher level of performance by simply changing settings in the Options Tool – for example, to require more than one success 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
by definition necessarily 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, or the student quits. The question is not whether a student will learn. The relevant questions are: a) how long learning will take and b) whether and how student can best be motivated to complete a given tutorial.

Prerequisites play an essential role in the process. Prerequisites correspond precisely to atomic (terminal) nodes in Flexform knowledge representations. Some prerequisites are so simple that they may (with given populations) safely be assumed on entry. For example, the ability to read and write numerals is commonly assumed by second grade. Entry with respect to other prerequisites, however, may be less certain. Before beginning a TutorIT tutorial for long division, for example, we would almost certainly want to make sure a student knows the multiplication tables and how to subtract. Similarly, no would want to teach column subtraction before a child knows the basic subtraction facts.

The basic question is how one make contact with learner’s who have not mastered such prerequisites? For example, how else can one teach column subtraction to a child who cannot write or recognize numerals (e.g., “5”, “3”)? SLT support for indefinite refinement offers a unique and systematic 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. SLT rules representing such 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 perform 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, for example, 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, the resulting analyses (for the same 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 the right key or combination thereof to get the solution. In effect, 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 on. 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). These six kinds of process 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 mandatory settings. Other options enable authors to more fully control the way content is delivered.

At the most basic level, 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, each student enters essentially as a blank slate – a “?” mark. 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.

Depending on prior student exposure to given content, for example, one might set “Start all Nodes with Learner Status” in AuthorIT’s Options Tool to “?” or “—“ (see Fig. 6). Teachers and tutors may select any number of other delivery options for individual (or all) students. TutorIT, for example, may start tutoring with each node
in the Learner Model (see below) marked 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 of 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 questions (or hints/scaffolding) even when the learner’s status on lower level nodes is unknown.

Authors have a wide variety of options making it possible to accommodate a wide variety of pedagogical biases – or should I say “instructional models”. Available options support any number 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. With the possible exception of overlooked bugs, they never make mistakes. Unlike other CBI (or ITS), however, TutorIT Math tutors are designed to ensure mastery. 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 in practice depends on whether:

- The associated SLT rule(s) accurately represents knowledge deemed important by subject matter experts and that this knowledge has been specified with sufficient precision (wherein terminals correspond to essential prerequisites),
- Students demonstrate mastery of prerequisites on entry and
- Students complete the TutorIT tutorial – the only way a student can complete such a tutorial is to demonstrate mastery on the skill being taught to whatever criterion the author has specified (in the Options Tool).

In effect, what students learn and whether or not a student who completes a tutorial actually learns a skill is not a question to be determined empirically. Rather, essentials are whether students are sufficiently motivated to complete a given tutorial, how long it takes, and generally what might be done to make the tutorial better (e.g., more efficient and/or motivating for students). Improvement will be incremental as feedback suggests and resources allow. Initial field trials are being being planned as this article goes to press.
Current tutorials focus on very specific skills. Guaranteed learning is restricted specifically to those skills. Nonetheless, many of the TutorIT tutorials developed to date also include instruction pertaining to meaning – the kinds of instruction commonly included in textbooks and/or classroom instruction.\(^4\)

There is no guarantee that students will necessarily master this supplemental material – material that is (or should be) included in classroom instruction. The question of whether and to what extent this supplemental instruction benefits students is an empirical one. In short, current TutorIT tutorials focus on what they can do better and more efficiently than a human. Meaning in current TutorIT tutorials is of secondary importance. They are designed to support classroom instruction by concentrating on those skills that are essential in all future learning. They are not designed to replace entire curricula or what a good teacher can (or should) do.\(^5\)

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. The way things are set up there is essentially no way a student can complete a TutorIT math tutorial without mastering the skill. The question is not learning as such but whether a student is motivated to complete a given tutorial, a very different question requiring a very different answer.

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

---

\(^4\) It is not that one could not target meaning as such. It is simply that doing so would require further analysis. For example, TutorIT Column Subtraction is based on a detailed analysis of what must be learned to perform column subtraction – with learning guaranteed when a student completes the tutorial. This tutorial also includes instruction describing and graphically illustrating a concrete model of what is being done step by step (e.g., when one borrows during subtraction). The difference is that we have not undertaken systematic analysis of what would need to be learned to ensure that a student is able to demonstrate the meaning associated with any given problem, or the reverse to construct a physical model corresponding to any given subtraction problem. Indeed, this is not normally done well in textbooks either. “Dienes blocks” developed in the 1960s by an old colleague of mine were designed precisely for this purpose.

\(^5\) While TutorIT Math is not sufficiently complete to cover everything in a typical textbook. Other than background reading and the like, it is an open question as to whether there are specific skills in a math textbook that could not be done as well (or better) in TutorIT.
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 highly 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: TutorIT represents a significant step forward in automation. As more and more tutorials are developed TutorIT can gradually take over tasks previously done well only 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 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. Automation is an on-going incremental process. Our children and our country will be the main beneficiaries.

4 CRITICAL ADVANCES IN CURRENT SLT THEORY

Analyzing Complex Domains

It might seem we are done! Given any domain, AuthorIT’s AutoBuilder component can be used 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.

It is very difficult, effectively impossible with many complex domains to represent what needs to be learned as a single integrated SLT rule (cf. Scandura, 1971, 2007). This is not simply a matter of having to compromise as regards completeness.

It is not possible to directly identify what must be learned to prove all known theorems in mathematics, or to specify how to write beautiful poems (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, limit precision – and complexity increases rapidly. More important, pedagogical decision making in both ITS and relational models depends inextricably on content semantics. At minimum, this dependence increases both development and evaluation costs.

In SLT, it might appear that one can avoid the problem of coverage by simply introducing a finite set of SLT rules. Given a problem domain, why limit oneself to a single SLT rule? One can always add new ones. Doing so, however, would not solve the fundamental problem. Given any non-trivial domain, it is impossible to directly identify everything a learner should know. This fact has been a central tenet in SLT from its inceptions (cf. Scandura, 1971a). It was the primary and original 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 minds. ITS knowledge engineers work with subject matter experts to identify sets of condition-action pairs, or productions representing relevant knowledge. Productions collectively are expected to be sufficient for solving arbitrary problems in some 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 current problem state and the goal. Mean-ends analysis seemed reasonable and gradually morphed into chain-
ing (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 (cf. Scandura, 2007).

A further limitation of learning mechanisms, as used in production systems, is that they impose essential constraints on implementation. Adding, removing or reordering individual productions may fundamentally effect behavior, but it will not fundamentally change how an ITS operates. Learning mechanisms, however, necessarily come “hard wired”. They cannot be added or removed without fundamentally effecting operation of a production system.

Recognizing this problem, 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 (CBM) in ITS development (e.g., Mitrovic & Ohlsson, 2007), where the focus is on constraints rather than productions. While CBM may reduce complexity, however, pedagogical decision making still depends on content semantics.

Quite independently, Polya’s (1960) early analyses of mathematical problem solving take a fundamentally different approach to the problem. Polya identified a number of domain specific “heuristics”, such as “the Pattern of Two Loci” and “Similar Figures”. He showed that heuristics play an essential role in various kinds of mathematical problem solving. Polya’s 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. Like learning mechanisms, however, they play a direct role in how new SLT rules are acquired and used.

\[ \text{NOTE: While influential in mathematics education, Polya’s (1960) work is not widely known in TICL circles.} \]

**SLT Solutions**

The SLT offers a detailed road map going forward. Consider the second and third advances mentioned earlier:

- 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.
- 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. In the next section, 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 in SA is on identifying 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 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. 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).

Historically, Structural (cognitive domain) Analysis (SA) has been used to systematically identify higher as well as lower order SLT rules. The process by which higher order SLT rules were constructed, however, has been largely subjective. This approach 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:

- Define a complex problem domain informally.
- Select a finite set of prototypic problems in that domain.
- Construct an SLT solution rule for solving each prototype problem.
- Construct a higher order SLT rule operating on other (typically simpler) SLT rules for constructing each SLT solution rule. (The SLT solution rules constructed are outputs in the range of the higher order SLT rule.)
- Eliminate redundant SLT rules (which can be derived by application of higher order rules to others).
- Repeat the process as desired, each time resulting in a set of SLT rules that are 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).

Earlier analysis of various complex domains (e.g., Scandura et al, 1974, Scandura, 1977, Scandura & Scandura, 1980) shows that two things happen as SA proceeds: The individual rules become simpler but the generating power of the rule set as a whole goes up dramatically, thereby expanding coverage in the 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 to date 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. Rule selection plays an essential role in motivation and problem solving in ill-defined domains where more than one SLT solution rule may be used (See Appendix A for other examples.).
Number Series Domain – consisting of sums of number series. (Step a above)

1. SME Selects Prototypic Problems (one of potentially many) (Step b above)

\[ 1 + 3 + 5 \rightarrow \text{sum} \]

2. Construct (multiple) SLT Solution Rules (2A, 2B, 2C) for Prototypic Problem (each rule can be refined where desired as above) (Step c above, but wherein each solution rule may systematically be refined as in Fig. 1 above)

\[
\begin{align*}
2A & : 1 + 3 + 5 \rightarrow 3 \times 3 \rightarrow 9 \\
2B & : 1 + 3 + 5 \rightarrow 3 \times (1 + 5)/2 \rightarrow 9 \\
2C & : 1 + 3 + 5 \rightarrow \text{successive addition} \rightarrow 9
\end{align*}
\]

3. Convert each SLT Rule into a Higher Order Problem (This is a critical new Step in identifying higher order SLT rules) (Construct Goal & Given of Higher Order Problem)

Higher Order Problem 3A (Given is on the first line below; Goal is on the second):

\[
\begin{align*}
1 + 3 + 5 & \rightarrow 3 \times 3 \rightarrow 9 \\
1 + 3 + 5 + 7 + \ldots & \rightarrow N \times N \rightarrow \text{Sum}
\end{align*}
\]

Higher Order Problem 3B:

\[
\begin{align*}
a + a + d + \ldots + L = a + (n - 1)d & \rightarrow 3(1 + 5)/2 \rightarrow 9 \\
a + a + d + \ldots + L = a + (n - 1)d & \rightarrow N(A + L)/2 \rightarrow \text{Sum}
\end{align*}
\]

Higher Order Problem 3C:

\[
\begin{align*}
1 + 3 + 5 & \rightarrow 1 + 3 + 5 \rightarrow 9 \\
a_1 + a_2 + a_3 + \ldots + a_n & \rightarrow \text{successive addition} \rightarrow \text{Sum}
\end{align*}
\]

4. Alternative SLT Higher Order Rules for Solving Higher Order Problems 3A, 3B, 3C (Step d above)

Higher Order Rule 3A: \( \rightarrow \text{replace 3 terms by } N \rightarrow \)
Higher Order Rule 3B: \( \rightarrow \text{replace 1 by } A, 5 \text{ by } L, \text{ 3 terms by } N \rightarrow \)
Higher Order Rule 3C: \( \rightarrow \text{replace each term by a variable, three terms by } n \rightarrow \)
The process of SA can be repeated (indefinitely). Step e (above) makes it possible (optionally) to eliminate redundant SLT rules. For example, the rule “4 × 4” for solving the problem “1 + 3 + 5 + 7” can be derived by applying higher order rule 3A, for example, to the number of terms, 4. In general, higher order rules make it possible to derive any number of new SLT rules from given ones.

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 → 50 × 50 → 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 in the number series domain. At this stage of SA, an analyst may eliminate redundant rules (as in Step e above) that may be derived (via other rules). Alternatively, deciding which SLT rule 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. Precisely the same principle applies in motivation. In this case, we are effectively considering a domain wherein students may do more than one thing (including ignoring the problem and doing something else).

In our example, the selection process represents a still higher order problem (so SA is repeated as in the original Step f). The given in the higher order 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. Rule selection has been an inherent part of SLT since its inceptions (cf. Scandura, 1971, 1973) but has not been systematically applied in application. Defining the domain structures associated with higher order SLT rules is essential. Data structures play a decisive role in determining which rules to use under what circumstances.
All higher order rules are domain dependent. While one can classify higher order rules by category, any given domain will invariably call for variations. Case based reasoning (CBR), for example, involves higher order rules that map solutions (SLT rules) for one kind of task into solutions for analogous ones (e.g., mapping counting up in addition to counting down in subtraction, or repeated addition in multiplication to repeated subtraction in division). While similar, higher order rules will to some extent depend on the domain from which they are derived.

(NOTE: Data structures in SLT correspond to assumed (automatically perceived) encodings and decodings, whereas procedures correspond to conscious operations.)

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, 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 may be viewed as a generalization of 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. Experience with Structural Analysis makes two additional 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, given the Journal of Experimental Psychology’s strong statistically oriented traditions, to get supporting deterministic research on UCM published in that journal (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 one above 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).

In retrospect, one can see why expert systems run into trouble. One reason is that knowledge engineering turned out to be very hard, slow and expensive, and that experts could not always articulate what they were doing. While Structural Analysis certainly does not trivialize the problem, it at least makes the task tractable. Recall in this context, the original hope that there were only a small number of basic learning mechanisms – preferably one. Alas, neither “means-ends analysis” as originally proposed (Newell & Simon, 1972), chaining nor any other mechanism proposed to date has satisfied that requirement.

SLT’s Universal Control Mechanism (UCM), on the other hand, does serve this role in unique fashion (Scandura, 2007; cf. Scandura, 1971a, 1973, 1974a,b). 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, explaining and predicting the behavior of individuals in specific instances is a distinguishing characteristic of 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 undue significance, 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 both the domain and the 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 support “chaining” although this technology 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 considerably further. Two objectives are on our near term technical agenda.

1. We will have to extend AuthorIT to support Structural (domain) Analysis (SA) in its entirety, including support for systematically representing higher order SLT rules.
2. We will have to formalize and implement SLT’s Universal Control Mechanism (UCM). UCM will enable TutorIT to generate solutions to ill-defined problems, an essential requirement for tutoring on ill-defined domains.

Specifically, 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, AutoBuilder does not currently support the construction of higher order SLT rules. Plans are in place for introducing automated support but this has not yet been realized.

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 extension requires replacing TutorIT’s current chaining mechanism with SLT’s Universal Control Mechanism (UCM). Fortunately, the current chaining mechanism is a separable module so its replacement and integration should be straightforward. The basic UCM design has been detailed in a recent patent. The main challenge is to implement, test and refine as necessary to ensure that UCM is ready for prime time.

I will not 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. 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.

Extension of AutoBuilder will in principle support Structural (domain) Analysis (SA) on any domain, no matter how complex. 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”. That is, the rules identified will be sufficient for generating solutions for what the analyst believes is a sufficient number and variety of problems in the given domain.

Armed with 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 also will 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 pattern. 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 a strictly 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 with higher and lower order SLT rules.

Given a representation of what needs to be learned, whether just lower order SLT rules as at present or including higher order knowledge as proposed, TutorIT can 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, as may be necessary in many complex domains. Even a small amount of analysis is better than little or none. Given their complexity, ITS do not have this flexibility.

**Comparison with Contemporary TICL Research**

Most TICL research today focuses on general models or frameworks. General frameworks represent a beginning, but I believe we can do more.

More attention in my view should be given to identifying as precisely as possible what needs to be learned for success. 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 more certain job one can do of teaching it. This holds whether one is talking about automated tutorials or human teachers. The main difference is that automation is more readily subject to incremental improvement.\(^6\) Compare the above with what currently is being done. Students often are placed in an immersive environment of one sort or another. Scaffolding (we used to call them “hints”) is typically indirect, or at best imprecise. Hints can certainly encourage, indeed require involvement of the learner. If successful, they also may exercise the learner’s cognitive abilities.

Unfortunately, systems of this type rarely if ever achieve 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, it should be possible to provide more targeted instruction.

---

\(^6\) Some might think we already know what needs to be learned in school math. Although analyses in school math tend to be more complete than in other areas, the analyses we have undertaken show that those used in planning textbooks, lessons, or instructional systems are invariably incomplete. It is not sufficient to simply list the kinds of problems to be solved, to name the particular skills required or even to identify all of the productions that might be involved in solution. Analysis must include identification of what needs to be learned at all meaningful levels of abstraction. Without full analysis, an automated tutorial will necessarily be incomplete, and cannot reliably guarantee mastery.
Encouraging learners to exercise whatever they may (or may not) know is a good thing. Nonetheless, two points need to be emphasized.

1. TutorIT can easily be configured to provide informal hints. There is nothing forcing TutorIT to be as precise as possible.
2. There will inevitably be learners for which typical, one size fits all scaffolding will be 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 AuthorIT and TutorIT would be capable of arbitrarily precise diagnosis and remediation on higher as well as lower order knowledge.

In this regard, I call attention to an early piece of research on math learning. In this research Roughead and Scandura (1968) were able to explicitly identify the higher order rules necessary for success in solving number series problems. 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 versus students who are taught those rules directly (e.g., by exposition) secondarily exercise and/or learn additional albeit potentially different skills. Students who discover higher order rules may in the process 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 do better than humans.

One further point deserves emphasis. TutorIT’s commitment to deterministic thinking (cf. Scandura, 1971, 2007) requires a significant change in how one goes about evaluating instruction. In particular, it calls into question the usual measures used in controlled experiments, and specifically the need for controlled experiments focusing on how much is learned. Well designed and suitably refined TutorIT tutorials build on what students already know and automatically adapt to individual needs during the course of (individualized) tutoring. By its very nature, TutorIT requires learners to demonstrate mastery of what is being taught. IF a learner enters with the necessary prerequisites (which can systematically be
AND completes such a tutorial that learner will necessarily have demonstrated mastery of what is being taught.

Evaluating automated instruction that adapts to individual needs, as does TutorIT, requires a different focus. Rather than comparing whether or how much various groups of students learn, the critical issues are whether or not a child is motivated to complete a given tutorial, how long it takes and how long the skill is remembered. Similarly, rather than comparing alternative incompletely defined treatments (e.g., TutorIT vs. classroom learning), one can definitively control and compare alternative delivery (i.e., tutoring) modes using TutorIT itself – without confounding content with methodologies.

Further Extensions

Although discussion is beyond the current scope, technologies based on SLT 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. Benefits in automatic problem solving also should be explored. For that matter, these ideas might reasonably be applied in areas as diverse as robotics and manufacturing.

Where do we go from here? Supporting complex domains will not come without a price. 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, it is possible and 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.

That said we have already developed a core set of TutorIT math skill tutorials covering the basic arithmetic and now beginning algebra. These tutorials represent only a beginning, but they do point the way toward a whole new generation of automated, highly adaptive tutorials. They also open heretofore-unavailable research opportunities, making it possible to better understand the benefits and limitations of various pedagogies.

TutorIT technologies have the potential of revolutionizing the way adaptive tutorials are developed, delivered and evaluated. They can be used to develop
highly adaptive tutorials in essentially any area: mathematics, reading, science or otherwise.

We can only do so much alone. Accordingly, I invite those of you who may be interested to join in the effort: Make others aware of TutorIT Math tutorials. Better, join us in future development. You can try TutorIT free by going to www.TutorITmath.com. If interested, feel free to contact me at scandura@scandura.com. In either case, it will be exciting to see the results of half a century of research making a difference in the real world.

REFERENCES

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.


### Software Engineering Publications


**Key Patent & Pending**


APPENDIX A

Steps 1–4 in Structural Analysis (SA) are illustrated above in the Number Series domain. Emphasis is given to steps 3 and 4 – which are essential in dealing with higher order knowledge. This Appendix illustrates SA with two other domains: Measure Conversion and Proofs in Trigonometry. Only top level SLT rules are shown below: Once a top level SLT rules has been constructed, refinement proceeds in the same way with all SLT rules, whether of 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

   \( \text{yd} \rightarrow 36 \_ \times \rightarrow \text{in} \)
   \( \text{gallons} \rightarrow 8 \_ \times \rightarrow \text{pints} \)

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

   \( \text{Givens:} \quad \text{yd} \rightarrow n_1 \_ \times \rightarrow \text{xxx} \)
   \( \quad \text{xxx} \rightarrow n_2 \_ \times \rightarrow \text{in} \)
   \( \text{Goal:} \quad \text{blug} \rightarrow n \_ \times \rightarrow \text{clug} \)

4. Construct SLT Higher Order Rule Composition Problem
   (Domain/Range Structure of H. O. Rule is Un-initialized Version of Higher Order Problem)

   \( \text{DOMAIN}*: \quad \text{blug} \ [n \_ \times] \ \text{xxx} \)
   \( \quad \text{xxx} \ [n \_ \times]) \ \text{clug} \)

CONSTRUCT PROCEDURE for Higher Order SLT (Composition) Rule

   \( \text{PROCEDURE: compose rules so output of first matches input to second} \)
Example 2: Proving Trigonometry Identities

1. SME Selects Prototypic Problem (one of many)

\[
sin^2A + cos^2A = 1 \quad - \quad ?\text{proof}
\]

2. Construct Solution Rules for Prototypic Problems

\[
sin^2A + cos^2A = 1 \quad \rightarrow \quad divide a^2 + b^2 = c^2 \text{ by } c, \text{ substitute sin, cos definitions} \rightarrow \quad \text{Proof is resulting steps}
\]

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

Given: \[sin^2A + cos^2A = 1 \quad \rightarrow \quad divide a^2 + b^2 = c^2 \text{ by } c, \text{ substitute sin, cos definitions} \rightarrow \quad \text{Proof is resulting steps}\]

Goal: \[\text{Trig Identity} \quad \rightarrow \quad divide a^2 + b^2 = c^2 \text{ by side, substitute trig. fn. definitions} \rightarrow \quad \text{Proof is resulting steps}\]

4. Construct H.O. SLT Generalization Rule

\[\rightarrow \text{Replace Specific Values (e.g., c, sin) with Generalizations} \rightarrow \]
\[(\text{e.g., } c \rightarrow \text{side}; \text{ sin } \rightarrow \text{trig_fns})\]

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 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 (in the main text) illus-
trates 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.