Knowledge Representation in Structural Learning Theory and Relationships to Adaptive Learning and Tutoring Systems

JOSEPH M. SCANDURA*

MERGE Research Institute

This article summarizes major steps in the evolution of Structural Learning Theory (SLT), a comprehensive, parsimonious, precise and operationally defined theory of complex human behavior. SLT covers knowledge representation, methods for constructing same, cognitive processes, knowledge assessment and interactions with external agents (e.g., teachers). The article emphasizes major advances in recent years that make full automation possible. It details and illustrates: a) ill-defined as well as well-defined knowledge, both represented in terms of SLT rules consisting of structural and procedural Abstract Syntax Trees (ASTs), b) how SLT rules can be represented at arbitrary levels of detail and how the higher as well as lower order SLT rules needed to master any given problem domain can be constructed systematically, c) cognitive mechanisms, including empirical data associated with a Universal Control Mechanism (UCM), which controls the use (and acquisition) of all SLT rules, subject only to a fixed processing capacity and speed constraints characteristic of individuals, d) how the knowledge available to any given individual (behavior potential) is operationally defined relative to SLT rules and e) theoretical, empirical and practical implications for building automated tutoring systems. The concluding section shows why the theory makes a difference, how it can be tested and what this implies for building e-learning, intelligent and other advanced tutoring systems.

Keywords: Structural Learning Theory, knowledge representation, cognitive theory, Abstract Syntax Trees, SLT rules, production systems, relational networks, control mechanisms, expertise, declarative knowledge, procedural knowledge, cognitive load theory, knowledge assessment.

*Corresponding author: scandura@scandura.com
My initial research on mathematical problem solving demonstrated that the timing of information had a greater effect on learning than whether that information was learned by discovery or exposition (Scandura, 1964a,b). Subsequent research reinforced the importance of what rules and higher order rules were being learned (e.g., Scandura, 1967, 1968, 1969; Scandura et al, 1967, 1968), and demonstrated that what is learned by any given individual can under carefully prescribed conditions be identified via single test items (cf. Greeno & Scandura, 1968; Scandura, 1969, 1970).

This research motivated development of the Structural Learning Theory (SLT), an attempt to identify fundamental assumptions from which these and other findings could be explained, predicted and indirectly controlled (via instruction). A good deal of research has gone on since SLT was first introduced (Scandura, 1971). The focus of this research, however, has always been on understanding fundamentals—on four basic questions:

- **Content:** What does it mean to know something? Specifically, how can competence (content knowledge) be represented so it has direct behavioral relevance (e.g., is executable)?
- **Assessing Behavior:** How can one determine individual knowledge? What does an individual know and not know with respect to any given content?
- **Cognition:** Why can some people solve problems whereas others cannot? What are the basic mechanisms and constraints governing how learners use and acquire knowledge?
- **Instruction:** How does knowledge change over time as a result of interacting with an external environment?

From its inceptions in 1970, SLT has aimed at and has largely achieved a high level of generality, coherence and parsimony. Parts of the theory were also

---

1 37 years have gone by since I first introduced the Structural Learning Theory (SLT) at the 1970 Structural Learning Conference in Philadelphia, repeated a couple days later at an invitational address to an American Educational Research Association audience, and later published in the Journal of Structural Learning under the title “Deterministic Theorizing in Structural Learning: Three Levels of Empiricism” (Scandura, 1971; becoming an ISI Citation Classic in 1978).

Although “structural learning” is probably more closely associated with me, the Journal of Structural Learning in which the article appeared was founded and edited by Z.P. Dienes, a well-known Hungarian mathematician and mathematics educator with a long historical legacy. Ironically, motivated by another of my earlier articles “The emerging field of psycho-mathematics” (Scandura, 1968), Dienes named an institute he founded in Sherbrooke, Canada, during the 1970s, “Center for Research in Psycho-mathematics.” Perhaps the best known albeit unfairly limited legacy of this pioneer in American mathematics education is the widespread use of “Dienes blocks” in helping children learn the meaning of place value in arithmetic.
sufficiently precise for implementation of certain essentials on a computer. What has been lacking is a way to represent knowledge in a way that fully captures (all) SLT essentials, an essential prerequisite to making it possible to systematically construct SLT-based instructional systems on a computer.

Initial attempts in SLT focused on representing lower and higher order knowledge in terms of Post-like production rules (e.g., Scandura, 1971; cf. top-level SLT rules below). The use of such rules was assumed to be governed by a “goal switching” control mechanism that played an essential role in determining what rule to use and when (Scandura, 1971, 1973).

As originally formulated, this control mechanism only applied to generating new knowledge (rules). Like “conflict resolution” in contemporary expert systems, rule selection originally required different mechanisms. Goal switching, however, was soon generalized to accommodate both knowledge generation and selection from among alternatives (cf. Scandura 1971 & Chapter 8, 1973). Unfortunately, it was not possible to completely separate this control mechanism from higher order knowledge (e.g. Wulfeck & Scandura, Chapter 14 in Scandura, 1977). Solution to this problem is detailed below.

The parallel need to represent individual differences—specifically, what individual learner’s know relative to what is to be learned—led to representation of SLT rules as directed graphs (equivalent to flowcharts) (Scandura, 1971, 1973, 1977). When the operations and decisions in directed graphs are atomic (universally known or unknown by all learners in the target population), research showed that it is possible both to identify what was known (and unknown) and to predict behavior with extraordinarily high degrees of precision (e.g., Durin & Scandura, 1974). What was missing from directed graphs, however, was a systematic way to represent expertise with respect to the same content. As detailed below, levels of expertise are a natural consequence of representing SLT rules in terms of Abstract Syntax Trees (ASTs).

This article details essentials of SLT as it stands today with a focus on how knowledge representation based on ASTs bridges the gap between high-level conceptualization and operational instructional systems. Equal attention is given to Structural (Domain) Analysis (SA), a method for systematically constructing knowledge representations along with a deterministic approach to knowledge assessment and cognitive theory, and their respective roles in building a broad range of teaching and learning systems. The article begins with a brief synopsis of the historical background leading to SLT.

These introductory sections are followed by:
• Representation of content knowledge (KR) in SLT in terms of Abstract Syntax Tree (AST) based SLT rules,
• Structural (domain) Analysis (SA), a systematic method for constructing such KRs,
• SLT's deterministic cognitive theory detailing how individual learners use and acquire knowledge,
• Assessment (operational definition) of individual knowledge in terms of the content knowledge to be acquired,
• Associated theory detailing how two (or more) individuals interact—with a focus on teaching and learning,
• Theoretical, empirical and practical implications—pulling it all together showing why the theory makes a difference, how it can be tested and what this implies for building automated tutorial systems,
• Current Status, Implications and Needed Research.

HISTORICAL BACKGROUND AND INTRODUCTION TO SLT

In the 1960 and 70s, and largely to this day, human behavior and psychological theories are assumed to be inherently probabilistic in nature. Rather than focusing on what individuals do in given situations, experiments typically focus on predicting the behavior of groups of people. It was surprising in that context that our early experiments showed that under clearly delineated circumstances it was possible both to assess what individuals knew and to predict what they would do in particular situations. This held both on simple tasks (e.g., Scandura, 1971; Durnin & Scandura, 1973) and in more novel problem solving involving explicitly defined higher order knowledge (e.g., Scandura, 1974; Wulfeck & Scandura, 1977).

---

I introduced Structural (Cognitive Task) Analysis in the late 1960s-early 1970s as an extension of task analysis. The latter at that time was concerned solely with behavior (cf. Robert Miller, 1959; Gagne, 1965). The original focus in Structural Analysis was on the cognitive competence/knowledge necessary for producing such behavior (Scandura, 1970, 1971, 1973; Durnin & Scandura, 1973), with special attention to higher order knowledge.

I use the term Structural (domain) Analysis herein to emphasize its focus on identifying knowledge needed to generate desired behavior rather than the cognitive processes learners actually use in solving given tasks. Theoretically, any given instance of behavior can be generated in any number of ways (e.g., as assumed in model tracing tutors, Ritter, 2005). With any given learner population, however, practice shows that relatively few methods are actually used or needed in instruction. As we shall see, no attempt is made to represent what individuals learn in an absolute sense but rather to identify what individuals know relative to what (one or more) experts believe should be learned to produce the desired behavior. This includes arbitrary kinds of higher as well as lower order knowledge.
Work in computer simulation during that period (e.g., Newell & Simon, 1972) was concerned with individual performance while solving problems. Rather than simulating human performance as a computer program, SLT was (and is) axiomatic in nature. SLT focused instead on identifying a small set of (deterministic) assumptions from which one could explain, predict and control the behavior of individuals in specific problem-solving situations. In this sense, SLT is similar to theories that dominated classical physics.

A sharp distinction was made from SLT’s inceptions (Scandura, 1971a) between a theory of competence (what needs to be learned), human behavior under idealized conditions (where the effects of memory are eliminated) and behavior where memory plays a direct role.

**Knowledge Representation**: The question of how to represent knowledge in information processing theories has a long history (e.g., Newell & Simon, 1972, Scandura, 1968, 1971, 1973). After a decade of early research on computer simulation of problem solving performance, for example, Newell and Simon (1972) finally settled on production systems, a formal executable representation developed by the logician Post in the 1930s. I also used production systems during this period, along with directed graphs, in my original work on SLT (1971, 1973, 1977). I finally settled on the latter because there was no convenient way to distinguish partial knowledge of productions.³

Another critical difference was SLT’s early emphasis on the rigorous identification and exploitation of higher order knowledge (e.g., referred to by Polya, 1960, as “heuristics”). Specific attention was given to SLT rules that operate on and/or generate classes of other rules. Higher order rules provide a general, parsimonious and potentially operational way to represent complex structured knowledge and to account for novel problem solving and creative behavior (e.g., Scandura, 1971b, 1973, 1974).

*(NOTE: Complex behavior in expert (production) systems is the result of combining individual productions via “hard-wired” control mechanisms such as*

³One certainly can make distinctions based on which productions in a production system (list) are known. Unlike SLT rules, however, individual productions are atomic (non-refinable) elements not designed to distinguish partial knowledge. SLT (content) rules represent to-be-acquired knowledge units used to generate problem solutions. SLT rules are not cognitive constructs (like productions) that learners are assumed to assemble according to prescribed controls and conflict resolution mechanisms. They are more general than productions in the sense that they represent both different levels of expertise and different degrees of knowledge at any given level of expertise. Productions effectively are special cases of (individual) SLT rules, which consist of single condition nodes and an indivisible operation (i.e., a condition-action pair).
means-ends analysis, chaining, and learning by example (cases) and conflict resolution mechanisms such as first or last, among others. Higher order rules associated with given problem domains serve this role in SLT.\(^4\)

The availability of higher order SLT (content) rules eliminates the need to anticipate how to solve all, or even most, problems in any given domain. Substantial research conducted under rigorous conditions, much of it over 30 years ago, demonstrates how novel problems can be solved by combining the effects of lower and higher order rules, the latter serving to generate new solution rules as needed (e.g., Roughead & Scandura, 1968; Scandura, 1971a, 1971b, 1973, 1974, 1977, Scandura & Scandura, 1980).

My early attempts to represent knowledge, however, were lacking in important respects, in particular how to represent levels of expertise. I explored alternatives ranging from production systems to category theory, including in-depth discussions with leading logicians, such as Dana Scott and Paul Rosenbloom (see Scandura, 1973, Chapter 2). I finally settled on the use of directed graphs (equivalent to flow charts) (e.g., Scandura, 1971, 1973) to represent (what I call SLT rules to distinguish them from productions). Unlike productions, directed graphs provide a convenient way to represent individual differences (as subgraphs).

Another aspect of Knowledge Representation (KR) that is often overlooked in information processing theories derives from the traditional distinction in experimental psychology between perception and cognition. The former deals with how the world is perceived by humans and the second, by conscious cognitive processing. Many psychologists believe that perception, while it may depend on prior learning, is a largely automatic process. Cognition, on the other, requires conscious processing. Moreover, as people gain expertise in an area they are able to process increasing amounts of information automatically.

Directed graphs (SLT rules), as such, fail to adequately capture different levels of expertise. Each level requires different assumptions as to what is atomic (what can be assumed as prerequisites available to targeted learners). In addition, there was no explicit provision in SLT rules for input-output structures (that determine domain of applicability)—only procedures.

\(^4\) Hard-wired mechanisms are represented in terms of higher order rules. As we shall see in the section on cognitive theory, there is only one universal control mechanism. This mechanism controls the use of individual SLT rules, including higher order rules providing the equivalent of chaining, generalization (from examples), analogy, etc. More generally, it determines both how new knowledge (rules) is created from existing rules and how conflicts are resolved when more than one rule is applicable.
Structures play a particularly important role in higher order knowledge, where over generalization is common.

**Need for Systematic Structural (domain) Analysis:** Given the wide variety of problem domains of interest in education (and elsewhere), it soon became apparent that having a way to represent knowledge would not be sufficient. Accordingly, considerable effort was given early on to developing a method for identifying the higher as well as lower order SLT rules required for success. Structural (Domain) Analysis (SA) was first used to identify higher and lower order SLT rules associated with a workbook (Scandura et al., 1971) to accompany a book I wrote on mathematics for elementary school math teachers (Scandura, 1971c). SA became increasingly systematic over time but until recently little or no provision was made for representing different levels of expertise, and the process of identifying higher order SLT rules relied heavily on analyst intuition.

As a result of complementary research in software engineering (e.g., Scandura, 1994a,b; US Patent 6,275,976-Scandura, 2001), it gradually became clear (Scandura, 2001, 2003) that there is a close and complimentary relationship between data structure analysis and process analysis. The more abstract the process, the more complex the structure and vice versa. The more detailed the process, the simpler the data. As we shall see, this observation plays a central role in representing different levels of expertise.

**Cognitive Theory:** Although essential, simply representing knowledge (as SLT rules or otherwise) is only a start. In order to build an automated learning or problem solving system, for example, one must introduce mechanisms to control how that knowledge is to be used and/or to resolve conflicts when more than one possibility exists. In this respect, cognitive theory in SLT is unique in its deterministic, yet highly flexible foundation. SLT’s cognitive theory derives from a small number of fundamental assumptions supported by key empirical results enabling deterministic prediction of individual behavior in particular situations (Scandura, 1971, 1973, 1977).

An important distinction is made between behavior in general and behavior under what I have called memory-free conditions, where the learner or problem solver is not encumbered by time or how much can be processed in memory at any one time. An open book exam provides one real world prototype. Another involves having unlimited time working with paper and pencil.

Accordingly, emphasis was given to testing fundamental mechanisms under conditions where the effects of incidental variables were eliminated. One key
assumption is that all knowledge (i.e., all higher and lower order SLT rules) are controlled by a single Universal Control Mechanism (UCM). Initially, UCM was loosely defined in terms of Goal Switching. If the learner has an SLT rule available for solving a given problem, then that rule (analogous to matching productions in a production system) would be applied. On the other hand, if no such solution rule is immediately available, the goal was assumed to switch to the higher level goal of deriving such a solution rule. UCM would then look for higher order rules that might be used in the derivation. Once identified, the higher order rule would be applied and the new SLT rule generated. Control was next assumed to revert to the original goal, where the new rule would be found and the problem solved (Scandura, 1971). This goal switching mechanism was later generalized to accommodate conflicts, where two or more rules might be used (Scandura, 1973, Chapter 8).

Testing in this context involves determining UCM's sufficiency, given any specific set of higher and lower order rules, in predicting problem-solving (on specific problems). Determining universal availability of UCM (rather than some undetermined and/or variable combination of SLT rules and UCM) requires ensuring that the learner has actually learned the specific rules in question (e.g., Scandura, 1971, 1973, 1974). Forgetting is limited as a possibility, if not eliminated entirely—e.g., by ensuring that needed SLT rules are available to the learner during testing. The rationale is exactly the same as when early physicists illuminated (i.e. minimized) the effects of friction by studying the rate of fall of various objects in a (near) vacuum. The basic methodology in both cases is clearly different from randomizing over unwanted conditions as is nearly universal in contemporary behavioral science.

Both SLT's memory-free and unrestrained cognitive theories are based on a series of foundational experiments that make it possible to explain, predict and allow behavioral control of individual behavior in specific situations. Rather than introducing probabilities, deviations from predication are treated as errors of measurement (e.g., Scandura, 1974). To make the point consider what might have happened in physics had Galileo in his thought experiments at the Leaning Tower of Pisa dropped an iron ball and a feather. Rather than Newtonian theory based on the assumption that all objects in a gravitational field fall at the same rate (mitigated only by friction and/or other forces—e.g., wind resistance), physics might instead have developed as a science of droppings—documenting the rates of fall of various kinds of objects. One can make the case for the equivalent in contemporary behavioral science.

As in early physics, theoretical predications in SLT are derived from a small number of fundamental assumptions—based on my conviction that behavioral
science is at a similar stage of development. Like physics, however, testing
detailed deterministic predications is not a routine task. This was a major
factor in introducing different levels of theorizing in SLT: a) Theories of
competence (what needs to be learned, as in linguistics for example), along
with a systematic method for constructing such competence theories. Such
theories are tested based on their ability to account for behavior in given
task/problem domains. b) Cognitive theory under idealized conditions
requires testing human (e.g. problem-solving) behavior in situations where
the effects of memory are eliminated. c) Testing human behavior in situations
where processing capacity and speed play a central role is still more
demanding. Even here, for example, ultra short term physiologically based
and/or transient effects (e.g., Sperling, 1960) may be eliminated (see

SLT’s unrestrained cognitive theory is highly comprehensive and cohesive. A
small core set of fundamental assumptions accommodate a broad range of
behavioral phenomena. Given the rules and higher order rules known to an
individual, SLT’s universal control and two fundamental processing constraints
(capacity & speed) provide an explicit basis for predicting behavior in a broad
range of specific situations (e.g., Ehrenpreis e & Scandura, 1974; Scandura et al,

**Instructional Theory:** SLT’s diagnostics provide an explicit operational
link between SLT rules identified via structural analysis and the knowledge
available to individuals. Specifically, they provide an explicit basis for
assessing what each learner does and does not know (i.e., the individual’s
behavior potential) at each point in time. Predictions can be tested with
arbitrary degrees of precision. Associated pedagogical logic determines the
instruction at each point in time.

The modularity of SLT rules makes both testing and instruction highly
efficient. Thus, while testing (and perhaps instruction) on a broad range of
problems will be desirable and the norm in practice, research suggests that
testing on small sets of foundational SLT rules may be sufficient (e.g.,
Scandura, 1974). The behavioral implications of knowing (all or part of) the
higher and lower order SLT rules (associated with any given problem domain)
collectively can be quite broad. Testing each such rule individually is
theoretically sufficient both for assessing behavior potential and for
determining needed instruction with respect to that domain.

Nonetheless, creating a rigorous executable foundation for automation was
extremely difficult. Although experimental support for the “goal switching”
mechanism, for example, was unusually strong (e.g., Scandura, 1971, 1973, 1974), we were unable in attempts at automation to completely separate the control mechanism from higher order knowledge (Wulfeck & Scandura, 1977).

This and other early attempts at formalization and automation failed to capture the full intent (of SLT). More important, local conditions at Penn moved SLT qua theory to the sidelines beginning in the later 1970s, even more so after Artificial Intelligence (AI) and a fledgling cognitive psychology exchanged concepts in establishing cognitive science. AI provided needed automation precision (via, e.g., production systems & recursive logic-inspired languages such as LISP and Prolog) and cognitive psychology provided results (e.g., limited processing capacity) derived from traditional experimental methods.

My emphasis during this period shifted to more immediately practical software research (e.g., building the first authoring system for the old Apple II computer in the later 1970s). We developed over 200 computer based learning systems ourselves, and licensees Borg Warner Educational Systems, Queue, Inc. and IQ-Ware developed several hundred more covering areas ranging from preschool to adult learning. The need for more sophisticated software in this environment led me increasingly toward basic work in software engineering leading to Flexsys, SoftBuilder and AutoBuilder, the latter providing the formal foundation for knowledge representation in SLT as it stands today.

**Relationships Between SLT and AuthorIT Technology:** Figure 1 adapted from Scandura (2005) shows a schematic overview of the role of Knowledge Representation in SLT and the AuthorIT technologies necessary for its implementation. As detailed therein parts of SLT have been implemented in an authoring system, called AuthorIT. Although AuthorIT’s design is extensible, two key parts of SLT have not yet been implemented: UCM and extension of AutoBuilder to accommodate higher order SLT rules. As we shall see, however, Knowledge (that which is to be learned) is represented as hierarchies of SLT rules, each SLT rule characterized as an operation defined on a hierarchical structure, called an **Abstract Syntax Tree (AST)**.

Knowledge available to individuals in the target population is characterized (measured) relative to these hierarchies of SLT rules. Individual knowledge corresponds roughly to a set of computer programs in which procedures are semantically meaningful (to human beings) operating on equally meaningful hierarchical data structures—although there are subtle but
very important differences. Among other things, we shall see that AST data structures consist of indefinitely refinable Domain (input) and Range (output) elements—not limited to “is a” and “part of” refinements. Procedures similarly consist of indefinitely refinable operations to be carried out.

(NOTE: These operations are similar to production rules but should not be confused with them because the entities on which they operate are SLT structures not relationships/conditions. It will nonetheless become increasingly clear below that while equivalent computationally, SLT rules have important advantages over the use of production rules in instructional systems.)

We shall also see that an SLT Rule hierarchy represents what is to be learned at multiple levels of abstraction (expertise). Each individual’s knowledge is defined by the highest level (most abstract) subset of SLT rules in each SLT rule hierarchy mastered by that individual. This subset of SLT rules can be represented formally as a directed graph in which the nodes are ASTs (cf., Scandura, 1971, 1973, 1977).

Directed graphs in the past have been used to represent individual knowledge (i.e. an individual’s behavior potential) (e.g., Scandura, 1971, 1973, 2001). As we shall see below, these sub-graphs represent single levels of expertise relative

---

FIGURE 1
Role of Knowledge Representation in SLT with relationships to the AuthorIT & TutorIT authoring & delivering systems used in its implementation.

---

KNOWLEDGE REPRESENTATION IN STRUCTURAL LEARNING THEORY 179
to given SLT rule hierarchies. We often use the term **SLT rule** also to refer to directed graphs where there is no chance of confusion.

The left side of the schematic represents **Structural (domain) Analysis (SA)**, which as we shall see includes a systematic way to represent hierarchies associated with each lower and higher order SLT Rule to be learned. We shall also see how SA makes it possible to refine each such hierarchy indefinitely. Each such hierarchy will be executable on a computer to the extent that terminal operations in the hierarchy are executable. The term “**higher order**” in Fig. 1 is underlined to highlight the fact that AuthorIT does not yet support SA with respect to higher order SLT rules. Accounting for broad and/or ill-defined domains further requires multiple interacting SLT rules, including higher order SLT rules that operate on and/or generate other SLT rules. We shall detail the necessary processes in the sections that follow.

The **Learner** in Fig. 1 is characterized in SLT by the individual’s knowledge along with a **Universal Control Mechanism (UCM)** and cognitive constraints pertaining to that individual’s characteristic processing capacity and speed. Unlike alternatives such as means-ends analysis, generalization, case-based reasoning, logical inference and the like, we shall see that UCM is completely independent of higher order knowledge. UCM is assumed to control the use and interactions among ALL Individual SLT rules, subject only to constraints imposed by each individual’s characteristic processing capacity and speed. As we shall see, experimental data and common observation support the assumption that processing capacity and speed are fixed for individuals (over relatively long periods of time). These constraints and UCM are operationally defined cognitive constructs (in SLT). Whether or not any have physical substance in underlying brain physiology is yet to be determined, and is not relevant for present purposes.

**TutorIT** in the schematic is characterized in SLT by the Universal Processing Mechanism (UCM), unlimited processing capacity and full knowledge of the content to be taught (represented by a set of SLT rule hierarchies) plus detailed knowledge of how to teach—how to diagnose what a learner does and does not know and/or what to teach at each stage of learning. **UCM** is highlighted to emphasize that it has not yet been implemented in TutorIT. We shall detail what needs to be done in the sections below.

Scandura (2001a, 2001b) provides an informal introduction, but omits essential detail necessary for implementation. The following sections are directed toward that end.
KNOWLEDGE REPRESENTATION IN SLT: DEFINITION OF SLT RULES IN TERMS OF ABSTRACT SYNTAX TREES (ASTs)

Representing knowledge in SLT is equivalent to representing what a subject matter expert (SME) believes must (or should) be learned to solve problems in a given problem domain. The subject matter expert is expected not only to identify SLT rules for solving sample problems in the domain but also to identify what must be learned to derive new SLT solution rules for solving novel problems in that domain and for representing that knowledge at multiple levels of expertise. This would be a tall order if the SME had to do this without automated assistance and/or rather specific guidelines. Fortunately, while currently incomplete, both exist and are sufficient for use in practice.

SLT Rules and Abstract Syntax Trees (ASTs): Like computer programs, SLT rules consist of a procedure and an associated data structure. Unlike traditional programs, however, the procedures and associated data structures directly represent the real world constructs to which they refer. Rather than trying to express an idea or process in terms of a given set of primitives (i.e., statements or routines in a standard programming language), processes in SLT rules are expressed in directly associated, easily understood semantics. Similarly, rather than data structures like int, char, arrays and records, all data structures in SLT rules are represented as ASTs in which all elements are typeless (actually node_pointers in AutoBuilder’s High Level Design [HLD] language).

Although ASTs are widely used in compiler theory and software engineering tools, data structures and processes have traditionally been analyzed independently. Information engineering focuses on organizing data hierarchically, whereas structured analysis (emphasis on ed to distinguish it from structural analysis, e.g., see SA below) focuses on process hierarchies. Each type of representation provides a different view of the intended target. In both cases, the basic idea is to represent software designs with increasing degrees of detail as one moves from high level design toward operating code. Software engineering tools typically represent information graphically, most commonly today using the Unified Modeling Language (UML). Unlike SA, however, there invariably is a gap between the lowest levels in any such design and actual code used to implement the design. In the next section we shall see that there is a close connection between structural and procedural refinement in SLT rule hierarchies.
Object orientation (OO) adds a further dimension (cf. Scandura, 2001). Rather than packaging operational resources (as in OO), associated behavior (e.g., cleaning different kinds of rooms) calls for abstractly similar operations (i.e., clean) on different structures (e.g., kitchen & bedroom objects). We shall also see that elements in structures may themselves be operations. This latter realization leads to a more precise characterization of higher order knowledge.

**SLT Rule Hierarchies:** Fig. 2 represents a hierarchy of SLT rules for column subtraction. Each node in this hierarchy represents an operation or condition, and the data structures on which it operates.

The top-level operation `subtract (1)` on the right operates on the (entire) `Prob` data structure (1) on the left. The operation `subt-c (2)` operates on the body of a loop, namely on the `Column` data structure (2) on the left. Similarly, the operation `subt-fact (3A)` and the condition `IF..THEN (4)` operate on still lower level data (e.g., pairs of digits).

---

**FIGURE 2**
Representation of structure (data) and procedure (process) ASTs showing corresponding structural and procedural refinement types.

Operation `draw (3B)` represents a third, essential kind of (interaction) refinement. This operation represents the top-level of an SLT rule that generates values of parameters (in this case, draws the `Difference`) in the parent operation.
FIGURE 3
The three procedural Flexforms at the top show different levels of refinement as they appear in AutoBuilder. The first Flexform shows only the top level of refinement (i.e. the top-level ColumnSubtraction operation and the Loop refinement. The second Flexform shows further refinement of the loop body. The third shows all levels, including a low level REPEAT..UNTIL loop corresponding to distinguishing '0's' in columns to the left. The bottom figure shows the data structure representing column subtraction problems represented as a Treeview. In general, Flexforms show as many or as few levels of abstraction as desired.
The three Flexforms at the top of Fig. 3 show successive levels of refinement of the same SLT rule hierarchy (except for 3B) in AuthorIT’s AutoBuilder component. The data structure at the bottom of Fig. 3 represents the associated data structure AST.

Notice that each individual operation in the SLT rule hierarchy generates a solution to a unique subproblem. The top-level operation `subtract` in Fig. 2 is equivalent to the top-level node `Column Subtraction (minus, underline: Prob;)` in Fig. 3. Both operate on a data structure called `Prob`. These top-level operations generate specific values of the difference digits in `Prob`, which are initially undetermined. Similarly, the operation `subtract_the_current_column (: Prob, CurrentColumn;)` in Fig. 3 takes partially solved problems as input and generates solutions to those sub-problems. That is, it generates the difference for a given column. Terminal operations, like `subt-fact` in Fig. 2 operate on individual digits. Condition nodes in the hierarchy compare digits (e.g., `TopGreaterThanBottom` just below the highlighted node in the middle Flexform).

*(NOTE: These operations are analogous to productions in production systems with two major exceptions: a) Conditions in productions are replaced by structure ASTs, and b) these structure ASTs define both the operation’s domain and range.)*

Whereas digits in column subtraction are normally viewed as atomic (indivisible), they may be further refined as shown in Fig. 2 into operations for reading or writing numerals (e.g., writing “5” using curved and straight line segments). More generally, our research demonstrates that ANY idea or process can be refined indefinitely into arbitrarily small data elements or processes (U.S. Patent 6,275,976, Scandura, 2001, 2003, 2005). In effect, prerequisites such as reading and writing numerals can always be made explicit by refining (i.e., defining) each as a separate SLT Rule.

Although column subtraction provides a simple and convenient example, it must be emphasized that exactly the same ideas apply in representing ANY content. In learning to read, for example, a child must (among other things) learn to create sounds corresponding to given words—e.g., given “bed” to say “b-e-d”. Prerequisites, in turn require learning and sequencing sounds associated with the letters in “bed”—for example, given “b” to say “bah”, given “e” to say “eh” and given “d” to say “dah”. Saying “b-e-d”, of course, makes no sense unless the child has first learned what that sound means. One can test for meaning by having the child say “b-e-d” given an actual bed, or a picture of same. Conversely, the child might be asked to select a picture of a bed (in a group of pictures) when told “b-e-d”. At the other end of the complexity spectrum one might start with a full text, and ask what a child must know to be able to read it, or (even) to describe what it means. The observant reader will recognize more than one way to do this, and the need for (higher order) selection. But let us not get ahead of ourselves.
Declarative Knowledge and Cognitive Models: Any number of programs can be defined on the same data structure (cf., database). Similarly, AST data structures can be viewed as declarative knowledge, on which any number of different SLT rules can be defined—the latter, in turn, make knowledge operational. For example, one can define SLT rules for addition, subtraction, multiplication, etc. by simply identifying specific node structures (on a common structure AST) to serve as domain (e.g., associated +, -, x and / signs) and range elements of associated computations (each represented as an SLT rule).

Cognitive Models can be represented similarly. For example, consider a simple cognitive map for finding one’s way in a simple grid of intersecting streets and avenues. Consider the following grid:

<table>
<thead>
<tr>
<th>Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avenues</td>
</tr>
<tr>
<td>A  B  C</td>
</tr>
<tr>
<td>Corner A-1&lt;sup&gt;st&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

Notice that each Corner has two parents, one Avenue and one Street. Any number of different SLT rules (and associated problems) can be defined on this Grid. Starting at Corner A-1<sup>st</sup>, assume the problem goal is to find the Corner C-3<sup>rd</sup>.

SLT rules used by people who remember directions by features along a route will be of the form,

“Start at Corner A-1<sup>st</sup>, go down Avenue A to Corner A-3<sup>rd</sup>, then turn left and go down Street 3 until Avenue C”.

People who prefer to use “cognitive maps”, will find equivalent alternatives—by thinking in terms of the Grid as a whole (rather than operating on individual corners). This sounds declarative until one realizes that this knowledge only becomes operational after one defines one or more procedures defined on that data structure. For example,

“I’m on Avenue A of the grid (which includes Streets 1, 2 & 3), and Street 1 (which includes Avenues A, B & C). I can get to my goal by going down either avenues or streets in any order as long as the letter or number keeps increasing.”

This operational knowledge can be represented either as a single SLT rule with corresponding decisions, or as a set of individual rules together with a
higher order selection rule for deciding which one to use in particular cases—but again let’s not get ahead of ourselves. Selection rules are considered below.

**SLT Rules—Representing What Must Be Learned**: The Flexform in Fig. 4 represents a SLT rule at a particular level of expertise. The operations and conditions in this Flexform represent a slice through the SLT rule hierarchy, corresponding to the order of execution. More generally, each SLT rule specifies a procedure operating on elements (data structures) in the corresponding structural AST. In Fig. 3 the operation *ColumnSubtraction*, for example, operates on *Prob*, which defines the entire data structure. This operation effectively defines an SLT rule for solving subtraction problems as a whole. At the next level, the operation *subtract_the_current_column* and the condition *NoMoreColumns* define a SLT solution rule operating on columns. The terminal operations and conditions in the full procedure shown in Fig. 3 define a SLT solution rule operating on pairs of digits.

In short, each SLT solution rule consists of a procedure together with the data on which it operates. The flow of control in any SLT rule is represented in AutoBuilder as a Flexform. SLT rules also can be represented as flowcharts or directed graphs (cf. Scandura, 1971, 1973, 1977 vs. 2005). Traditionally, the operations and conditions in flowcharts are atomic. That is, they are assumed to be available to all learners in the target population. As we shall see in the section on Assessing Behavior Potential, atomic representation makes it possible to assess the ability of individuals to solve associated problems (cf. Scandura,
It does not, however, make it possible to represent different levels of expertise.

Representing knowledge as hierarchies of SLT operations (degenerate rules) provides an explicit basis for representing entire classes of directed graphs (i.e., SLT rules), each representing equivalent knowledge at a different level of abstraction. As shown in Fig. 4, SLT solution rules can be viewed as slices through an SLT rule hierarchy. Each slice may involve any number of combinations of operations and conditions in the corresponding hierarchy, and represents a specific level of expertise. Where there is little chance of confusion, the term in SLT rule is used irrespective of whether one is referring to an entire slice or to a single operation.

Relationships Between Cognition and Observable Real World Problems:
Representing SLT Rules in terms of ASTs has the advantage of representing important aspects of encoding (perception) and decoding as well as cognition. SLT rules not only represent knowledge but also play a critical role in how humans interact with the real world. The AST structure associated with an SLT rule is assumed to define how the external observable world is understood and
acted upon (cf. McIntyre, 2000). Structural ASTs in SLT rules impose a corresponding structure on the problems to which they apply. The more precisely defined (refined) the data structure in an SLT Rule, the more detailed the problem structure. The learner sees only what he or she knows. Procedural knowledge comes into play only after a problem’s structure has been encoded.

Fig. 5 shows how the observable subtraction problem shown in the center panel in AuthorIT’s Blackboard Editor component is defined by the data structure in the left panel. (The gray squares correspond to-be-determined output variables.) In effect, domain (input) structures determine how observable problems are (perceptually) encoded. Range (output) structures define cognitive elements to be decoded (made observable). Both encoding and decoding are assumed to be largely automatic. Once learned, such processes are often hard (without explicit analysis or training) for the “knower” (now expert) to describe—as Bruner (1956) once said, “it is often hard to recapture conceptual innocence”.

**Higher Order SLT Rules—Generation, Selection and Chunking:**
Terminology commonly used in the literature on higher order knowledge, such as meta-knowledge, strategies, heuristics, analysis, generalization, abstraction, etc., tend to be ambiguous and leave considerable room for interpretation. We use the term higher order SLT rule uniformly. In all cases, domain elements in associated data structures include other SLT rules. Range elements include SLT rules generated by procedures in the higher order rules. As we shall see in the next section higher order SLT rules depend on and are derived from the domain in question.

In general, higher order SLT rules may operate on other SLT rules in three basic ways:

1. **Higher order SLT rules may generate new SLT rules that are not in the original rule set:** Higher order SLT generation rules are used to acquire new rules as needed, for example, to solve novel problems where no SLT solution rule is directly available (in the original rule set).

   Higher order SLT rules correspond to hardwired mechanisms in machine learning systems. All such mechanisms, whether they involve chaining, analogy, generalization, case-based reasoning or whatever, are all represented in SLT as learnable higher order rules.

2. **Higher order SLT rules may select from among alternative SLT rules:** Higher order SLT selection rules operate on two or more SLT rules and
select the one to use. Such rules play an especially important in solving design problems. Houses, for example, can be built in any number of ways. As any architect knows the challenge is to decide what kind of house to build and how. Experts in essentially every realm face similar decisions.

Higher order selection rules correspond in expert systems to hardwired “conflict resolution” mechanisms used to select from among two or more matching productions.

3. **Higher order SLT automation (or chunking) rules construct higher level (not higher order) SLT rules in SLT rule hierarchies from behaviorally equivalent lower level SLT rules:** Higher order SLT automation rules operate on SLT rules at one level in one or more SLT rule hierarchies and generate equivalent SLT rules at a higher level of abstraction. For example, subtraction problems may be solved by working column by column or by simply “knowing” extended facts like 50 -34. Higher order SLT automation rules in this case operate on less efficient SLT rules (e.g., working column by column) and generate more efficient ones. One such higher order SLT rule, for example, might shortcut the regrouping process. That is, the higher order SLT rule might modify the regrouping process—e.g., taking the complement of the ones digit (replacing the 4 with the 6) and one less than the difference between the tens columns (5—3) - 1 = 1). Higher order rules of this sort apply to similar problems, and may be useful with addition as well (for example).

Higher order rules of this type have the potential of detailing what happens during “chunking”. Rather than simply attributing “chunking” to practice, analysts have the option of detailing higher order SLT “chunking” rules—which would make the process explicit (and teachable).

As detailed above, different levels of expertise correspond directly to different levels of procedural abstraction in an SLT rule hierarchy. The procedures in this hierarchy correspond to behaviorally equivalent SLT rules in which the data structures become increasingly complex as one ascends the hierarchy. Higher order SLT “chunking” rules in this context map lower-level, more detailed SLT rules into higher-level SLT rules which operate on more complex structures. Accordingly, they provide an explicit mechanism detailing how procedural knowledge, where procedures are relatively complex and data structures simple, is converted into declarative knowledge, where procedures are simple and data structures relatively complex.

Like all higher order SLT rules, these “chunking” rules may operate between abstraction levels in any number of SLT rule hierarchies. Very little explicit research
has been done in this area but the potential implications for promoting expertise would appear to make such research well worth the effort (cf. Fadde, 2006).

*(NOTE: Parent-child mappings introduced in Scandura (2003) to ensure consistency between levels in SLT rule hierarchies are higher order SLT “chunking” rules restricted to individual hierarchies.)*

As we shall see in the section on cognitive theory, only one Universal Control Mechanism (UCM) is needed in SLT. This mechanism controls the use of all SLT rules, including all kinds of higher order rules. In the table below, chaining, analogy, generalization from examples (case based reasoning), etc, as well as “chunking” and rule selection (as in conflict resolution) represent various kinds of higher order SLT rules (denoted $\rightarrow\rightarrow$ vs. $\rightarrow$ for lower order rules). In general, higher order rules in SLT may involve arbitrary combinations of these kinds of higher order rules. As we shall see, this Universal Control Mechanism (UCM) represents a least common denominator, controlling the use of all SLT rules, whether higher order or lower order, in all learning situations—whether this involves “chunking”, deriving solutions to novel problems or selecting from among alternatives (as in design problems where more than one SLT rule may be used).

**Schematics for Various Kinds of Higher Order SLT rules**

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition</td>
<td>$A \rightarrow B, B \rightarrow C \rightarrow \rightarrow A \rightarrow (B \rightarrow) C$</td>
<td></td>
</tr>
<tr>
<td>Analogy</td>
<td>$A_1 \rightarrow B_1 \rightarrow \rightarrow A_2 \rightarrow B_2$</td>
<td></td>
</tr>
<tr>
<td>Generalization</td>
<td>$A_0 \rightarrow B_0 \rightarrow \rightarrow A \rightarrow B$</td>
<td></td>
</tr>
<tr>
<td>Selection</td>
<td>$A_1 \rightarrow B_1, A_2 \rightarrow B_2 \rightarrow \rightarrow A_1 \rightarrow B_1$ or $A_2 \rightarrow B_2$</td>
<td></td>
</tr>
<tr>
<td>Automation</td>
<td>$A_1 \rightarrow B_1, A_2 \rightarrow B_2 \rightarrow \rightarrow A \rightarrow B$</td>
<td>where $A$ = parent of $A_1, A_2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$B$ = parent of $B_1, B_2$</td>
</tr>
</tbody>
</table>

*NOTE: These schematics represent the kernel of truth in scenarios used in Case Based Reasoning (CBR, e.g. Kolodner et al., 2007), and various problem typologies (cf. Polya, 1962; Chapter I in Scandura, 1971c; Jonassen, Spector & others).*

**Individual Knowledge**: Another important distinction in SLT is made between knowledge that is to be learned and what an individual actually knows
and can do. The former, sometimes called **content knowledge** corresponds to competence as the term was originally used in linguistic theories (e.g., Chomsky, 1968; cf. Scandura, 1968a,b). **Individual knowledge** represents what the learner knows, and in SLT the potential behavior of which the individual is capable.

Given some domain of interest, whether it be broad, like a natural language, or a narrowly defined topic in arithmetic, content knowledge or competence refers to that which makes it possible to generate solutions to tasks or problems associated with that domain. **Content knowledge** is represented in terms of SLT rules—each consisting of a procedure and the AST data structures on which it operates.

SLT rules are implemented in AutoBuilder’s extensible and easily understood AST-based High Level Design (HLD) language (see www.scandura.com). Each SLT rule in a SLT rule hierarchy represents equivalent knowledge at a different level of abstraction—where each level of abstraction corresponds to a different level of expertise.

As above, each SLT rule represents a single procedural slice through a SLT rule hierarchy. Procedures at different abstraction levels in the hierarchy represent different levels of mastery: Procedures in **declarative knowledge** are relatively simple (e.g., *ColumnSubtraction*). Structures are correspondingly complex (i.e., *Prob*). Declarative knowledge corresponds to expert knowledge, where the procedures used to generate answers are relatively automatic (i.e., do not require execution of a more detailed procedure). On the other hand, procedures in **procedural knowledge** are correspondingly complex (e.g. composed of potentially numerous low-level operations and conditions). The structures are relatively simple (e.g., individual digits). In short, knowledge is not simply declarative or procedural.

In effect, whereas declarative and procedural knowledge are commonly distinguished in the cognitive sciences, all knowledge in SLT includes both in varying degrees. SLT rules may be simple or complex depending directly on complexity of the subject matter and inversely on the sophistication of the intended population of learners.

Moreover, lower order and higher order SLT content rules may be domain dependent and/or domain independent. Meta-knowledge/ higher order knowledge/ heuristics/ logical inference are simply different names for SLT rules in which one or more elements in the Structure AST is itself an SLT rule.

**Individual knowledge** (i.e., an individual’s behavior potential) is represented relative to SLT content rules. If the knowledge associated with a problem domain consists of a set of higher and lower order rules, individual knowledge relative each of those rules can be determined independently. Individual knowledge is discussed further and operationally defined in the section on **Assessing Behavior Potential**.
Individual knowledge can be understood in two very different ways. Originally, each individual’s behavior potential was represented with respect to specific SLT rules at particular levels of abstraction, each consisting of a non-degenerate procedure and its associated data structures (cf. Scandura, 1971; 1973, Chapter 10; 1977, Chapter 7). This method was easy to understand. Individual knowledge was simply represented in terms of known and unknown paths through the procedure (directed graph) used to represent the to-be-acquired knowledge. It also provides an explicit basis for instruction (cf. Scandura, 1971; 1973 Chapter 7; 1977, Chapters 8-14).

Unfortunately, this method of assessment also had an important disadvantage. Given a path there was no way in general to automatically identify the class of problems associated with this path. Whether or not a subtraction problem involves borrowing, for example, can be determined directly by inspecting the problem. This is not the case, however, in long division where the adequacy or inadequacy of the trial divisor depends on decisions made while solving a problem. An expert analyst can always identify problems associated with a given path by careful analysis of the subject matter—but this imposes an important in principle barrier to automation.

Assessing behavior potential with respect to SLT rule hierarchies not only solves this problem but makes the entire process even more efficient. In particular, individual knowledge may be represented in terms of nodes (single operations and conditions) in a SLT rule hierarchy. As we shall see in the section on Assessing Behavior Potential, each such node automatically generates both a sub problem associated with any given problem schema, and its solution.

(Nota: This approach may seem obliquely familiar to those working with production systems because individual rules in a SLT rule hierarchy involve a single operation. SLT rule hierarchies differ fundamentally, however, in the sense that the SLT rules therein are arranged hierarchically—representing equivalent behavior at different levels of expertise (e.g., consider a person knowing a top-level SLT rule versus a behaviorally equivalent lower level SLT rule).

In effect, the learner model may be represented either in terms of paths through given SLT content rules as done originally, or as an overlay on an SLT rule hierarchy (cf. Scandura, 1971, 2005). Both provide a sound, sufficient and highly efficient basis for specifying diagnostic and tutorial logic that is completely independent of content semantics. The original approach of representing SLT rules
as directed graphs offers an easily visualized measuring device—with individual knowledge represented as subgraphs. As we shall see, however, individual knowledge also can be represented as simple overlays on SLT rule hierarchies.

The latter (current) approach has the added advantage of being fully operational in the sense that it makes it possible to automatically both generate and solve problems associated with arbitrary nodes in an SLT rule hierarchy—depending solely on the structure of the nodes in the (procedural) hierarchy.

As we shall see, distinguishing behaviorally equivalent SLT rules at different levels of abstraction also makes it possible to identify higher order SLT automation (i.e., chunking) rules in a form they can be taught and learned (cf. Scandura, 2003).

Relationships between SLT rules, SLT rule hierarchies, SLT content rules and individual SLT rules are summarized in Table 1.

TABLE 1
Terminology Associated with SLT Rules.

<table>
<thead>
<tr>
<th>SLT Rule Hierarchy</th>
<th>hierarchy of SLT rules (i.e., operations and conditions with associated AST data structures)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLT Content Rule</td>
<td>procedure &amp; its associated AST data structure (analogous to a program)</td>
</tr>
<tr>
<td>Individual SLT Rule</td>
<td>represented either as a slice (directed graph) through an SLT Rule Hierarchy or as a learner model represented as an overlay on an SLT rule hierarchy</td>
</tr>
<tr>
<td>Other Terms Associated with Knowledge and SLT Rules</td>
<td>mastery level: represented by SLT rules at specific levels of abstraction in a SLT rule hierarchy</td>
</tr>
<tr>
<td></td>
<td><em>declarative knowledge: Procedure is relatively simple (e.g., top-level); Data structures are correspondingly complex (e.g., defined by the top-level).</em></td>
</tr>
<tr>
<td></td>
<td>*procedural knowledge: Procedure is relatively complex (low-level); Data structures are correspondingly simple (i.e., low-level).</td>
</tr>
<tr>
<td></td>
<td>higher order knowledge/meta-knowledge/heuristics/inference: Structure of SLT (higher order) Rule includes other SLT Rules. For example, higher order rules generate new SLT rules</td>
</tr>
<tr>
<td></td>
<td>conflict resolution/rule selection/design alternatives: Higher order rules select from alternative rules (e.g., as in solving design problems)</td>
</tr>
<tr>
<td></td>
<td>automation: Higher order SLT chunking rules map lower level SLT Rules to behaviorally equivalent higher level SLT Rules</td>
</tr>
</tbody>
</table>
| *Note: There are multiple gradations between declarative and procedural knowledge as well as variations on and combinations of the above schematics for higher order rules.
In the next section on **Structural (Domain) Analysis (SA)**, we detail an explicit method by which the SLT rules and higher order rules associated with given problem domains can be derived systematically. Given any problem domain, SA makes it possible to systematically derive both SLT rule hierarchies from high level of abstractions and higher order relationships—all in fully operational form.

**STRUCTURAL (DOMAIN) ANALYSIS (SA)**

Knowledge representations in Intelligent Tutoring Systems (ITS) are commonly constructed by a process called knowledge engineering. Subject Matter Experts (SME) identify knowledge needed to solve problems in the given domain. SME are typically aided by KR experts who convert this knowledge into executables (e.g., productions), sometimes with the aid of tools (e.g., Shute et al, 1999) and sometimes not.

In contrast, the process of Structural (domain) Analysis (SA) in SLT derives from task analysis (e.g., Robert Miller, 1959; Gagne, 1965). Task analysis focuses on to-be-acquired behavior. In addition to behavior, SA puts the emphasis on what must be learned, including higher order knowledge. Our early research on rule learning (e.g., Scandura et al, 1967; Scandura, 1967, 1968; 1969a, 1969b, 1970; Scandura & Durnin, 1968; Roughhead & Scandura, 1968) showed that detailing expected behavior and what must be learned to produce that behavior made empirical research largely redundant. Research aimed at identifying higher order knowledge in mathematics had the same result (e.g., Roughhead & Scandura, 1968, Scandura, 1971a,b). The single most important conclusion of this research is that the more precisely one identifies what learners need to know to produce desired behavior, the less important confirming experimental research becomes.

Given any task, it is always possible to identify what needs to be learned with some degree of specificity. On the other hand, examples of both research and practice where this is NOT done are not only commonplace, but the norm. I believe this lack of attention to what must be learned is a major source of the often conflicting, tenuous and/or weak results obtained in a good deal of contemporary educational research.

This research naturally led to development of a systematic method for identifying what must be learned for success in any given domain. SA was first used to identify the higher (as well as lower) order knowledge underlying a mathematics textbook for teachers (Scandura, 1971c). This analysis served as
the foundation for writing an accompanying workbook (Scandura et al., 1971d) covering over 500 lower and higher order rules. The results of subsequent research by Ehrenpreis and Scandura (1974) in which subsets of these rules were taught explicitly showed that: teaching higher order rules not only (a) eliminated the need to teach lower order rules explicitly, but also (b) enabled learners (elementary school teachers) to solve novel problems that those in the control group were not. They were taught less but learned more.

SA was later refined and applied in a number of research studies involving higher mathematics (Scandura, 1973), straight-edge and compass construction problems in geometry (Scandura et al., 1974; 1977, Chapter 3), algebraic proofs (Scandura, 1977, Chapter 4, with Durnin) and Piagetian Conservation (Scandura & Scandura, 1980). Given any problem domain, whether narrow and highly prescribed in scope or broad with informally defined bounds, SA has evolved into a highly systematic and now patented method (U.S. Patent 6,275,976, Scandura, 2001) for identifying lower and higher order SLT rules sufficient for solving sufficiently broad classes of problems in any given domain.

**Introduction to Structural (domain) Analysis (SA):** Recent advances (in SA) are a direct result of representing knowledge (what is to be learned) as hierarchies of SLT rules, including higher order SLT rules in which the data structures include other SLT rules. The top level of abstraction in every SLT rule consists simply of names of the associated operation along with inputs and outputs in the associated data structure—just a bit more specific than naming categories in problem typologies (e.g., Jonassen, 2007). In AuthorIT’s High Level Design (HLD) language such abstractions are represented in the form

```
operation_name (input: input-output; output)
```

An important advance in SA was recognizing that any SLT rule can be detailed to an arbitrary degree by refining a top level abstraction successively into components, categories and/or dynamic processes (Scandura, 2003, 2007). Data structures (concepts) and processes (procedures) can both be refined systematically and indefinitely via these three refinement types, and variants thereof (e.g., prototype refinements). Each such refinement can be carried out in a highly consistent manner so that the parent in each refinement is behaviorally equivalent to its children (see US patent 6,275,976, 2001; Scandura, 2003 for details).

*(NOTE: Indefinite refinement presents an enormous opportunity in building advanced learning and tutoring systems. The indefinite refinement hypothesis makes it possible to represent ANY content as an executable. Finding counter*
examples, real world examples where the above refinement types are not sufficient, poses a deep theoretical challenge to the TICL community.)

Two of these refinement types, component and category, are familiar in all hierarchical representations. **Component** refinements correspond to “IS A” relationships, or elements of a set (e.g., columns in a subtraction problem). **Category** refinements correspond to “PART OF” relationships or subsets of a set (e.g., columns that require borrowing and columns that do not).

All hierarchical representations inevitably result in (non-unitary) relationships (e.g., 9 is greater than 7). The problem with such relationships is that there is no uniform method for further refinement (as there is with components and categories). The mathematical equivalence between (dynamic) operations and (static) relations provides a way out. For example, the relationship “9 is greater than 7” can be represented as an operation (e.g., counting or one-to-one matching). **Dynamic** refinements correspond to relational refinements, the difference being the emphasis on construction (e.g., constructing the numeral ‘5’ from straight and curved line segments) versus static relationships (i.e., between straight and curved line segments).

**Dynamic** refinements play an essential role in ensuring full hierarchical representation. Whereas there is no uniform way to refine relationships into more elementary units, the introduction of dynamic refinements makes it possible to continue refinement indefinitely.

The key is to represent knowledge so that each level of abstraction is behaviorally equivalent to every other level (Scandura, 2001 [U.S. Patent 6,275,976], 2003, 2005). **Indefinite** refinement ensures that any given SLT rule can be made atomic— that is, can be refined to the point where each terminal SLT rule in the hierarchy is sufficiently simple, and thereby uniformly available to each individual in any given learner population.

**Indefinite** refinement also assures that any and all content, if sufficiently analyzed, can be made executable. Given a problem, executable content ensures that solutions can be generated automatically—thereby alleviating authors of that responsibility. As we shall see, representing to-be-acquired content knowledge as SLT rule (AST) hierarchies provides a sufficient basis for automating both diagnostic and instructional processes associated with that content. Among other things, hierarchical representation makes it possible to construct general-purpose diagnostic and instructional systems that are completely independent of content semantics (cf. Scandura, 2005).

**Relationship Between Structural and Procedural Refinement**: There is a direct relationship between structural and procedural refinements (e.g., see Table [Table 196 SCANDURA])...
2 in Scandura (2005) [p. 208]). As we shall see below, data elements in a component refinement can be carried out independently. Accordingly, the corresponding procedural refinement is a parallel refinement. A special case of component refinement is a prototype refinement where the number of components may vary—as in the number of columns in a subtraction problem. The corresponding procedural form is a loop.

Category refinements refer to data elements, which differ as to either value and/or structure. In the former case, the corresponding procedural refinement is selection refinement (e.g., IF..THEN or CASE). In the latter case, the corresponding procedural refinement is what I’ve (Scandura, 2003) called an abstract operation in which parameters refer to classes of data elements (where the structure differs as well). An abstract operation for adding numbers provides a simple example. Adding in general includes special cases of whole numbers, integers, rationals (e.g., decimals, fractions) and irrationals, each of which requires special treatment. How one cleans a room similarly depends on the type of room (bedroom, kitchen, etc.). This idea is closely related to Object Oriented design in which child elements have a distinctive structure while sharing certain elements in common with their parent.

Dynamic refinements involve defining (i.e., further refining) a parent data node as an operation with its own inputs and outputs. The resulting operation, in turn, can be refined as with any other SLT rule. Corresponding interaction refinements in procedural refinements involve refining a parameter of an operation. A dynamic parameter is a parameter of an operation whose value may vary independently of and feed back into that parent operation (Scandura, 2005). In Windows programming, dynamic parameters correspond to what are called callbacks. The numerals (e.g., 5, 6, etc.) used in performing arithmetic calculations are dynamic in this sense. Correspondingly, the procedures (SLT rules) used in constructing such numerals interact with (are operated on by) the addition, subtraction, etc. operations acting on them.

Discovery of the close relationship between structural (concept/data) refinement types and their procedural equivalents represents an important advance in knowledge representation. Not only does it allow arbitrary refinement (one need not get stuck with irreducible relations), but it also provides an explicit foundation for the equivalence between structural (declarative) and procedural knowledge. The only difference between expert and neophyte knowledge in this KR (and the gradations between these two extremes) is how much is structural and how much is procedural. As we shall see below, acquiring expertise in this context involves the gradual conversion of procedural into structural knowledge.
In effect, variations on three basic types of refinement are sufficient to refine any given data structure indefinitely: **component** [element of], **category** [subset of] and **dynamic** [construction of]. Moreover, each type of structural refinement corresponds to a type of process refinement: **parallel**, **selection** and **interaction** (e.g., callback).

**Declarative versus Procedural Knowledge—Levels of Expertise:** As above, each kind of structural refinement corresponds directly to a particular kind of procedural refinement. Figure 3, for example, shows that the prototype refinement involving current_column corresponds to the procedural loop through the various columns in any given subtraction problem. Distinguishing columns according to whether the top digit is greater than the bottom corresponds to the selection (IF..THEN) refinement shown in the procedural Flexform. Further refinement involves the presence or absence of a “0” at the top of the next column. In short, each type of structural refinement corresponds to a corresponding type of procedural refinement (see Table 2 in Scandura, 2005).

Top-level rules in an SLT rule hierarchy operate on top level, complex structures in the data AST. (Full) knowledge of subtraction at this level, for example corresponds to expert or declarative knowledge: “You give me the problem and I will give you the answer immediately—no dilly dallying with internal processes”. The expert is able to respond quickly (essentially automatically) to any given subtraction problem. The procedure used is assumed to be internal and immaterial. The associated individual knowledge may be assumed to be atomic, or equivalently perceptual in nature. Idiot savants (among others) automatically compute numbers in this manner. Furthermore, all arithmetically literate adults (and many children) know numerous shortcuts—e.g., for extended facts like 50 - 25. The latter correspond to intermediate level procedures operating on intermediate structures. Conversely, knowledge associated with lower-level SLT rules is correspondingly more procedural in nature.

**Higher Order Knowledge:** Representing knowledge as sets of SLT rules provides a concrete foundation not only for representing alternative levels of expertise but also for representing the knowledge associated with arbitrarily complex problem domains. Identifying higher order knowledge needed in discovery learning or solving novel problems, for example, is done in essentially the same way. The inputs and/or outputs in SLT rules may themselves be actions or operations (cf. Scandura, 1974). As we saw in the last section, Selection and even Automation or “Chunking” rules are formally the same (as derivation rules). The only difference in the case of higher order selection rules is that they
act on two or more available rules and select one for use. One commonly used selection rule, for example, is to select a rule that immediately generates a solution to a problem (i.e., when one already knows the answer) even when one also knows a general but more complex procedure for solving the problem. Higher order automation rules are similarly assumed to be responsible for creating shortcuts and the like (see above).

A major limitation of SA is that the process of identifying higher order SLT rules has required a deep insight on the part of the Subject Matter Expert. Converting solution procedures into higher order problems (see below) remained largely informal until recently.

Structural (domain) Analysis (SA): A Systematic, Extensible & Patented Method for Subject Matter Experts (SME) to Represent Observable Behavior & Knowledge as AST-based Problems & SLT Rules: With this background, we now turn to the task of showing how one can systematically construct a finite set of SLT Rules associated with any given problem domain—specifically, a set of SLT rules represented at arbitrary levels of expertise that collectively are sufficient for generating solutions to sufficiently broad classes of problems in that domain. Structural (domain) Analysis (SA) is a systematic method for constructing sets of SLT rules associated with any given content domain.

TABLE 2

Steps in Structural (domain) Analysis (SA)

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

As shown in Table 2, SA starts (1) with a subject matter expert (SME) selecting a set of prototypic problems in the target domain. The SME then (2)
systematically constructs SLT solution rules for each prototype problem, representing what is to be learned at multiple levels of abstraction. The previous section shows how both perceptual and cognitive knowledge can be represented at multiple levels of abstraction via only three basic refinement types—**component, category and dynamic**—and corresponding types of procedural refinement—**parallel, selection/object oriented & interaction** (also see Scandura, US Patent 6,275,976, 2001a,b; 2003, 2005). Starting from the highest-level abstraction, the SME need progress only as far as he or she wants.

The structural AST (in the SLT Rule) imposes a corresponding AST structure on problems. Fig. 5 shows a subtraction problem in AuthorIT’s Blackboard Editor. For example, the Top digits (nodes) in the tree view (at the left) under the hundreds, tens and ones place values in Fig. 5 have been assigned values 8, 7, 9, respectively. (The top digits are covered by answer boxes and do not show in the center panel.) Similarly, the Bottom digits have the values 5, 4 and 3. Initializing input nodes corresponding to prototype refinements consists of specifying a specific number of structures (columns) in the prototype. The prototype current_column in Fig. 4 represents the common structure shared by the hundreds, tens and ones columns. In effect, whereas prototype refinements represent a common structure (e.g., of each column), each specific problem has a fixed number of such structures (i.e., hundreds, tens, ones).

The next step (3) is crucial in dealing with larger ill-defined problem domains where each solution rule is systematically converted into a higher order problem (see section on Analyzing (SA of) Simple Ill-Defined Domains). Fourth (4), a higher order solution rule is constructed for each higher order problem. Redundant solution rules (which can be derived via the higher order rules) are optionally eliminated in step 5. The process can be repeated (6) with higher order rules as desired until sufficient coverage of the domain has been achieved. Research demonstrates that individual rules become simpler as the process proceeds but that coverage expands dramatically. Coverage in this case refers to the extent to which solutions can be generated to unanticipated problems in informally defined domains.

**SA of Simple Well-Defined Problem Domains:**— A domain is well defined if all problems in the domain fall into one of a (usually small) finite number of problem types, each of which can be characterized (solved) by a single SLT Rule. As above, each SLT content rule can be represented with any desired degree of precision.

Steps 1 & 2 result in hierarchies of SLT rules. The SLT rules in each such hierarchy operate on data at different levels of abstraction. Each of these SLT
rules generates equivalent behavior but with differing degrees of efficiency. They also differ as to the mix between procedural and structural complexity. Procedures in procedural knowledge are relatively lower level and more complex operating on relatively simple data structures. Conversely, procedures in declarative knowledge are relatively simple, but operate on correspondingly complex data structures. Multiple gradations between top-level and terminal level procedures support arbitrary mixes (of declarative and procedural knowledge).

**Step 1. SME Selects a finite sample of problems which fully characterize the domain.** Computational arithmetic, for example, may be characterized in terms of sample problems for addition, subtraction, multiplication and division of whole numbers. One might add fractions, decimal arithmetic, even verbal problem solving, etc. The domain of interest is strictly optional, and can be extended at any point in SA.

**Whole Number Arithmetic**

\[
\begin{array}{c}
4027 \\ - 2535 \\ + 37
\end{array}
\begin{array}{c}
324 \\ 256 \\ 0
\end{array}
\begin{array}{c}
324 \\ x 37
\end{array}
\begin{array}{c}
37 \\ 285
\end{array}
\]

One SLT solution rule is sufficient to solve each problem type: one each for (whole number) addition, subtraction, multiplication and division.

A very different problem domain involves cleaning bedrooms. Whatever the domain, the SME attempts to select problems that exercise the full scope of the solution rules he has in mind. The problem depicted in the picture below with initial and goal values fully exercises the solution procedure detailed in Step 2 below.
Step 2. SME systematically constructs a hierarchy of SLT solution rules—a hierarchy of procedures and corresponding data structures for each prototype problem. SA of arithmetic computation offers little new. We have already illustrated essentials for subtraction. In this section, we detail Step 2 in SA with an SLT rule for cleaning a bedroom. Arrow 1 points to the top-level SLT rule

\[\text{clean\_room} (\text{: bedroom;})\]

In this example, we define \textit{bedroom} as consisting of two components, \textit{bed} and \textit{carpeting}. AutoBuilder automatically assigns the names \textit{clean\_room\_bed} and \textit{clean\_room\_carpeting} to the child operations.

\textit{(NOTE: These names can be changed as desired to reflect the intended semantics.)}

Because the top level \textit{bedroom} structure (arrow 1) is refined into two components, \textit{bed} and \textit{carpeting}, the SLT rule \textit{clean\_room} is refined into two operations which may be carried out in parallel:

\[\text{clean\_room\_bed} (\text{: bed;})\] and \[\text{clean\_room\_carpeting} (\text{: carpeting;})\]

2. Systematically Construct Hierarchical Set of SLT Rules for Clean Room

FIGURE 6A
SLT rules for cleaning a room at different levels of abstraction.
As shown in Fig. 6B, carpeting (arrow 2) is further refined into the prototype current_rug (arrow 3). Hence, the corresponding procedural refinement in Fig. 6A is a REPEAT UNTIL loop, with each iteration operating on current_rug. current_rug can be clean, dirty or messy&dirty as shown in the picture.

(NOTE: Fig. 6B is incomplete suggesting that current_rug can only be clean or dirty.)

Each of these cases must be dealt with separately. Accordingly, the SLT rule representing the body of the loop clean_room_carpeting_alias (an automatically generated name that can be changed at will) is refined into the three cases pointed to by arrows 3A, 3B & 3C in Fig. 6A.

![SLT Clean Room Rules Reference Data at Different Levels of Abstraction](image)

In effect, Fig. 6A shows successive levels of refinement of SLT rules. Fig. 6B makes the structures on which those rules operate explicit (as ASTs). The essential point is that any SLT rule can be refined arbitrarily to reflect any desired degree of precision.

(NOTE: AutoBuilder is fully aware of the relationships between structural and procedural refinements. Given a structural refinement, for example, it automatically suggests the corresponding procedural refinement.)

To summarize Steps 1 and 2, the SME first selects a sample of problems, which fully characterize the domain. A “good” prototype will exercise all-
important characteristics of the SMEs preferred solution rule. If the goal were a procedure for subtracting whole numbers, then the problem might involve regrouping across zeros, regrouping and no regrouping. Similarly, in adding fractions, it would be desirable for the prototype to involve mixed fractions, involving different denominators.

Initially, these problems may simply consist of rough paper and pencil descriptions. The important point is that the SME believes that these problems adequately represent the domain. Although not essential, a reasonable next step is for the SME to sketch a solution procedure for each prototypic problem. The SME would then use AutoBuilder systematically, starting by selecting an appropriate name for the (solution) operation, including names for its domain and range parameters. It is always possible to construct a top-level SLT rule for any given problem, no matter how complex or incompletely defined.

With this as a starting point each data element and operation may be refined successively as far as is desirable and possible given available time and other resources. One never gets deadlocked with irreducible relationships. Ideally, analysis continues until contact is made with assumed prerequisites—in which case terminals are referred to as “atomic”. Whether atomic or otherwise, SLT rules are sufficient for defining sub problems and their solutions (from problem schemas). Making terminals executable in AuthorIT is a task for HLD programmers.

(NOTE: Once terminals in a SLT rule hierarchy (represented as a Flexform in AuthorIT) have been implemented, each and every Flexform node becomes an executable with its own AST data structure. This is a general property of abstract syntax trees—e.g., as used in compiler theory.)

Structural and/or procedural refinement may be carried out systematically in either order. The SLT rule hierarchy represents the associated knowledge at multiple levels of abstraction. The structural AST effectively defines the problems formally represented in AuthorIT’s Blackboard Editor.

(NOTE: We’ve seen previously that individual digits, which are normally viewed as static (e.g., in computational algorithms) can be refined further into SLT rules for constructing same. Any SLT rule that references such digits effectively operates on other rules. By definition, it then becomes a higher order rule. We now turn to a second (the original) way in which higher order rules naturally arise in SA.

**SA of Simple Ill-Defined Domains, with emphasis on identifying SLT solution Rules & Higher Order Rules:** We illustrate the process with three different examples: Measure Conversion, Proving Basic Trigonometric Identities and Number Series.
Example 1: Measure Conversion

The SME begins as always by selecting a representative set of problems—i.e., intuitively different problems that require different kinds of solution methods. The goal is to select intuitively different problems, which the SME believes will exercise most if not all of the solution methods he or she wants learners to know. Since emphasis here is higher order rules (rather than hierarchies), it is sufficient initially to represent problems informally. Given information in each of the following problems is in blue on the left. The problem goal is after the ? in red on the right.

Steps 1 & 2: Steps one and two follow as before.
1. SME Selects Prototypic Problems
   
   3 yd — ?in
   
   2 gallons—?pints

2. Construct SLT Solution Rules for Prototypic Problems
   
   \( \text{yd} \rightarrow 36\_\text{times} \rightarrow \text{in} \)
   
   \( \text{gallons} \rightarrow 8\_\text{times} \rightarrow \text{pints} \)

Steps 3 & 4: Step 3 is critical in analyzing complex or otherwise ill-defined domains. This step requires the SME to construct the Goal and Given defining a Higher Order Problem whose solution is the SLT solution Rule in Step 2, and whose Givens are in (and/or should) be in the set of SLT rules characterizing the given problem domain.

Step 3 requires two things:

A. Replacing semantic-specific nodes in the SLT Solution Rule with abstractions, supersets and/or objects.*

B. Selecting otherwise available rules from which the SLT Solution Rule can be constructed.

*(NOTE: The following examples all involve converting elements into sets. It is assumed, however, each of the three basic refinement types has an inverse that may be used to identify higher order problems: component \( \rightarrow \) abstraction, category \( \rightarrow \) superset, dynamic \( \rightarrow \) object.)*

Goals in a higher order problem are constructed by converting one or more constants in a solution rule into variables. For example,
yd \rightarrow 36\_times \rightarrow \text{in}

is converted into

blug \rightarrow n\_times \rightarrow \text{clug}

by replacing \( yd \) with \( \text{blug} \) (a made up name for an arbitrary measure unit), \( \text{in} \) with \( \text{clug} \) and/or and \( 36\_times \) by \( n\_times \). Obviously, the more constants converted to variables, the more general will be the goal, and accordingly the higher order problem. In this example:

- \( \text{blug} \) & \( \text{clug} \) represent generalized units of measurement. They correspond to abstractions is a structural AST.
- \( n \) represents any specific number.

In general, the givens in the higher order problem serve as data—in this case a pair of simple measure conversion rules, which when appropriately combined satisfy the higher order goal. Variations on the above include substituting “\( op \)” for “\( times \)”. This would extend the range of applicability, but it would also raise the question of whether this would lead to overgeneralization—leading to errors. These judgments must be left to the SME.

Convert SLT (solution) Rule to Higher Order Problem

**Construct Goal & Given of Higher Order Problem**

Givens: \quad \begin{align*}
yd & \rightarrow n\_1\_times \rightarrow xxx \\
xxx & \rightarrow n\_2\_times \rightarrow \text{in}
\end{align*}

Goal: \quad \begin{align*}
\text{blug} & \rightarrow n\_times \rightarrow \text{clug}
\end{align*}

Step 4 is exactly as before. The domain, range and procedure of the higher order SLT rule are constructed by starting at the top and refining as desired. In this case the procedure corresponds to a partially refined SLT content rule.

Variations on the above include substituting “\( op \)” for “\( times \)” in step three. This would extend the range of applicability of the SLT rule in step four, but it would also raise the question of whether this would lead to overgeneralization—leading to errors.

4. Construct SLT Higher Order Rule Composition Problem

**Domain/Range Structure of Higher Order Rule is an Un-initialized**
(General) Version of the Higher Order Problem) and

**DOMAIN:**
- $\text{blug} \rightarrow \text{n\_times} \rightarrow \text{xxx}$
- $\text{xxx} \rightarrow \text{n\_times} \rightarrow \text{clug}$

**RANGE:**
- $\text{blug} \rightarrow \text{n\_times} \rightarrow \text{clug}$

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

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

**Example 2: Proving Trigonometry Identities**

**Steps 1 & 2:** Although the subject matter is quite different, steps one and two remain essentially the same. Again, this SLT content solution rule is described at an intermediate level.

1. **SME Selects Prototypic Problems**
   - $\sin^2 A + \cos^2 A = 1$ — ?proof
   - $\cot^2 A + 1 = \csc^2 A$ — ?proof
   - $\tan^2 A + 1 = \sec^2 A$ — ?Proof

2. **Construct Solution Rules for Prototypic Problems**
   - $\sin^2 A + \cos^2 A = 1$
   - $\rightarrow$ start with $a^2 + b^2 = c^2$, divide by $c$, substitute $\sin, \cos$ definitions
   - Proof is resulting steps

**Steps 3 & 4:** Again, converting the solution rule in step 2 to a higher order problem involves: (1) identifying given SLT rules (and/or other data) from which the original solution rule can be constructed and (2) replacing specifics in the original solution rule with abstractions. In this example, the given of a higher order problem is the SLT rule for proving the identity $\sin^2 + \cos^2 = 1$ and the goal is an SLT rule for proving similar trig identities. Instead of dividing by the hypotenuse (side $c$) of a right triangle and $\sin/\cos$, we would use whatever side and trig functions are relevant in the given problem. We simply replace the given identity with an arbitrary identity in trigonometry. Realistically, of course, these identities would be limited to those involving the standard trig functions $\sin, \cos, \tan$, etc., which involve ratios of sides $(a, b, c)$ in right triangles.

3. **Convert SLT Rule to Higher Order Problem**
(Replace Semantic-specific Nodes in Solution Rule With Abstractions & Select Given Rule from which a Higher Order SLT Rule can be Constructed)

\[ \sin^2 A + \cos^2 A = 1 \]

→ start with \( a^2 + b^2 = c^2 \), divide by \( c \), substitute \( \sin, \cos \) definitions
  
  Proof is resulting steps

Trig Identity

→ start with \( a^2 + b^2 = c^2 \), divide by side, substitute \( \text{trig. fn.} \) Definitions
  
  Proof is resulting steps

The higher order SLT would replace specifics in a given procedure with generalizations (e.g., instead of dividing by \( c \), divide by whatever side is relevant). The advantage of such a rule is that it is very general and would apply to rules having scope well beyond simple trigonometry. Its disadvantage is that it would be easy to overgeneralize. Proper scoping can only be obtained via detailed analysis of the structures involved. It should be pointed out, however, that even this simple high-level analysis goes further than what is normally done in teaching trigonometry. Further analysis would only make the SLT rules more precise, and instruction more efficient.

4. Construct Higher Order SLT Generalization Rule

→ Replace Specifics (e.g., \( c, \sin \)) with Generalizations
  (e.g., \( c \rightarrow \text{side}; \sin \rightarrow \text{trig. fn.} \))

Example 3: Number Series

Steps 1 & 2: This example illustrates what happens when there is more than one SLT solution rule.

1. SME Selects Prototypic Problem

   \[ 1 + 3 + 5 \quad \rightarrow \quad ?\text{sum} \]
   \[ 1 + 3 + 5 + \ldots + 99 \quad \rightarrow \quad ?\text{sum} \]
   \[ 2 + 5 + 8 + \ldots + 32 \quad \rightarrow \quad ?\text{sum} \]
   \[ 3 + 5 + 5 + \ldots + 23 \quad \rightarrow \quad ?\text{sum} \]

For present purposes we identify three SLT solution rules for the first prototypic problem.
2. Construct (multiple) Solution Rules for Prototypic Problem

\[ 1 + 3 + 5 \rightarrow 3 \times 3 \rightarrow 9 \]
\[ 1 + 3 + 5 \rightarrow 3 \times (1+5)/2 \rightarrow 9 \]
\[ 1 + 3 + 5 \rightarrow \text{successive addition} \rightarrow 9 \]

**Step 3:** In this case we generalize each of the three SLT solution rules in Step 2. Notice that each of these generalizations (shown in red) results in a different SLT rule.

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

**SLT Rule 3A:**

<table>
<thead>
<tr>
<th>Given</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 + 3 + 5 )</td>
<td>( 3 \times 3 \rightarrow 9 )</td>
</tr>
<tr>
<td>( 1 + 3 + 5 + 7 + \ldots )</td>
<td>( n \times n \rightarrow \text{Sum} )</td>
</tr>
</tbody>
</table>

**SLT Rule 3B:**

<table>
<thead>
<tr>
<th>Given</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 + 3 + 5 )</td>
<td>( 3 \times (1+5)/2 \rightarrow 9 )</td>
</tr>
<tr>
<td>( a + a+d + \ldots + L=a+(n-1)d )</td>
<td>( n(a+L)/2 \rightarrow \text{Sum} )</td>
</tr>
</tbody>
</table>

**SLT Rule 3C:**

<table>
<thead>
<tr>
<th>Given</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 + 3 + 5 )</td>
<td>( 1+3+5 \rightarrow 9 )</td>
</tr>
<tr>
<td>( a_1 + a_2 + a_3 + \ldots + a_n )</td>
<td>( \text{successive addition} \rightarrow \text{Sum} )</td>
</tr>
</tbody>
</table>

Where \( n \) = no. terms

\( a/L/d = \text{first/last term/common difference} \)

\( a_i = \text{arbitrary term} j = 1, 2, 3, \ldots \) in arithmetic series

**Step 4:** As shown below, each of these higher order problems results in a different solution rule.

4. Alternative SLT Higher Order Rules to Solve
Alternative Higher Order Generalization Problems

**SLT Rule 3A:**

<table>
<thead>
<tr>
<th>Given</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 + 3 + 5 )</td>
<td>( 3 \times 3 \rightarrow 9 )</td>
</tr>
<tr>
<td>( 1 + 3 + 5 + 7 + \ldots )</td>
<td>( n \times n \rightarrow \text{Sum} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow \text{replace 3 terms by } n \rightarrow )</td>
</tr>
</tbody>
</table>
SLT Rule 3B:

\[
\begin{align*}
1 + 3 + 5 & \rightarrow 3(1+5)/2 \rightarrow 9 \\
\text{Goal} & \quad \text{Goal} \\
\text{Given} & \quad \text{Given} \\
\text{Procedure} & \rightarrow \text{replace 1 by a, 5 by L, 3 terms by n} \\
\end{align*}
\]

\[
\begin{align*}
a + a+d + \ldots + L = a+(n-1)d & \rightarrow n(a+L)/2 \rightarrow \text{Sum} \\
\text{Goal} & \quad \text{Goal} \\
\text{Procedure} & \rightarrow \text{replace 1 by a, 5 by L, 3 terms by n} \\
\end{align*}
\]

SLT Rule 3C:

\[
\begin{align*}
1 + 3 + 5 & \rightarrow 1+3+5 \rightarrow 9 \\
\text{Goal} & \quad \text{Goal} \\
\text{Given} & \quad \text{Given} \\
\text{Procedure} & \rightarrow \text{replace each term by a variable, three terms by n} \\
\end{align*}
\]

The higher order SLT generalization rules in examples 3A, 3B and 3C result in different generalizations, each with a different domain of applicability (see Scandura, Woodward & Lee, 1967; Scandura, 1967). Although structural analysis had not been developed at that time, the rules derived by subjects in these studies are precisely those indicated above.

**Rule 3A:** Replacing the number of terms by \( n \) & multiply \( n \times n \), is very efficient but works only with arithmetic series beginning with 1 with a common difference of 2.

**Rule 3B:** Replacing the number of terms by \( n \), the first by \( a \), the last by \( L \) and computing \( n(a+L)/2 \) is not quite as efficient but it works with ALL arithmetic series.

**Rule 3C:** Replacing each term by a variable & adding successively is very inefficient but it works with ALL series, arithmetic or otherwise.

This is exactly the kind of situation one faces in a typical design problem. There will normally be multiple ways to proceed. The question is what is the best way under given circumstances. We can only make progress in this direction by giving more attention to the identification of higher order selection rules. The following suggests how one might proceed.

**Step 5. SME Eliminates Redundant Solution Rules.** Given any problem domain, SA results in a set of SLT rules, including both higher order SLT rules and lower order ones. Rules that can be generated by the interaction of higher and lower order rules in the rule set become redundant. Such SLT rules can be derived as needed and, as we shall see, need not be taught directly.

Higher order rule(s) identified in Step 4 typically make it possible to generate solutions for any number of problems of similar type. In Example 1 new conversion rules can be generated as needed from a much smaller set of basic
rules (e.g., 1 ft. = 12 in., 4 qt. = 1 gal., etc.). Moreover, new (basic) rules can be added at will. Availability of a higher order for combining (chaining) basic rules effectively eliminates the need for rules that can be derived from combinations of higher and lower order rules. Such rules become redundant and may (optionally) be eliminated by the SME (as optional for instructional purposes).

Results of a cited study by Ehrenpreis & Scandura (1974) demonstrated that explicit instruction on higher order rules can increase transfer while dramatically reducing the number of rules that need to be learned. One can literally teach less AND have students learn more.

It also is important to emphasize that higher order rules may be misapplied—hence the importance of identifying each SLT rule’s domain of applicability. SA as defined above is an important step in this direction. For example, it would provide a more rigorous and complete characterization of Polya’s heuristics (1960) for mathematical problem solving than results reported earlier (Scandura et al, 1971d, 1974; Scandura, 1977, Chapters 3 &4). What was missing in those early analyses was the domain of applicability.

In Structural Analysis, as currently defined, domain of applicability becomes increasingly explicit as one refines higher order SLT rules. Accordingly, one can view problem solving typologies as very high level representations of SLT rules (cf. Jonassen, 2007, Spector, 2006). The challenge in each case is to define the associated ASTs more precisely, thereby enabling more definitive experiments and associated application to instruction.

**Step 6. The SME can continue the process indefinitely.** Example 3 is illustrative. We have three different SLT rules that might be used to solve various number series problems. How is a learner to decide which one to use? The fact that the process of SA can be continued indefinitely provides a solution to this problem. Newly identified (higher order) rules, for example, can be converted into still higher order problems. Example 3 provides an instructive example of how Higher Order Selection (aka Conflict Resolution) Rules arise naturally during the course of Structural Analysis (SA). Specifically, repeating steps three and four (in SA) results in the identification of higher order SLT selection rules.

*(NOTE: Higher order SLT selection rules correspond directly to what are called Conflict Resolution mechanisms in production systems. Rather than being hardwired in SLT; however; selection rules are strictly modular and may be added, modified and/or deleted without affecting other SLT rules.)*

Consider two simple higher order selection rules. The first higher order selection rule is guaranteed to work with all arithmetic series. In so far as it goes, it is optimal in the sense that it uses simpler SLT rules whenever possible.
6A. Case Type-of-Series:
   a. starts with 1 & has a common difference of 2: select rule N^2
   b. common difference: select rule N(A+L)/2
   c. default: select successive addition

The following selection rule is quite general but error prone. It works generally but it is not hard to find a counter example.

6B. Choose the simplest rule

Clearly, the first selection rule (6A) will work with any number series of the form
\[ a_1 + a_2 + a_3 + \ldots + a_{n-1} \]
whereas the latter selection rule (6B) will not (always work).

Learning German—A Final Example: The above examples may lead to the false impression that Structural Analysis is only useful with highly structured content such as school mathematics, and possibly science (aka STEM). The following example in learning German (of which the author knows little) demonstrates that SA can be used far more broadly—furthermore, that even a crude SA serves a useful purpose in identifying what must be learned. In this case, we represent the to-be-acquired knowledge schematically: [brackets] on the left designate an idea to be described, ?<German Phrase> designates a to be constructed German description of the given idea and arrows (→operation→) bracket a procedure for constructing that description.

Consider saying something like “Unfortunately, I can only speak a little German.”. We can represent the problem as:

[know little German]—?<German Phrase>

A naïve learner’s knowledge base, like my own, might consist of something like the following set of lower order SLT rules,

[I] → ich, [German] → Deutch, [a little] → ein wenig, [Unfortunately] → leider, speak → sprechen, [can] → kann, [only] → nur,

Together with the higher order SLT rule:
The neophyte learner will have presumably learned something like the following solution rule, which for economy of space is represented without successive levels of refinement simply as:

\[
[\text{know little German}] \rightarrow \text{“Leider, Ich kann nur ein wenig Deutsch sprechen”}
\]

Someone who knows German well (an expert) is able to also express the same thought in any number of different ways.

\[
[\text{know little German}] \rightarrow \text{“Ich bin im Deutschen ein Anfänger”},
\]
\[
[\text{know little German}] \rightarrow \text{“…”}
\]

To express a thought in acceptable German, the naïve learner must systematically apply the higher order SLT rule. The neophyte has effectively memorized the statement and can express it when the situation demands. The expert, on the other hand, as in the number series example above, must make a choice. This choice will invariably be made based on various, presumably subtle, cues—much like my young grandson, Sam. Unlike his PopPop, Sam is a natural born politician who from age two on, meeting a stranger for the first time instinctively and properly distinguished between such greetings as “Hi” and “Hello”.

(NOTE: These cues had nothing to do with whether or not the stranger was another child.)

**Summary:** Analysis of several fairly complex domains (e.g., Scandura et al, 1974, Scandura, 1977, Scandura & Scandura, 1980) shows as SA proceeds that two things tend to happen: The individual rules become simpler but the generating power of the rules set as a whole goes up dramatically, thereby expanding coverage in the original domain.

What was not done in this early research was to give explicit attention to domains of applicability (i.e., to structural ASTs). This is one place where more research needs to be directed. The higher order rule schematics shown above are commonly acknowledged in the literature, albeit typically as hard wired control
or conflict resolution mechanisms. In SLT, such schematics provide only a starting point, each identifying a type of higher order rule. The more precisely each can be detailed (i.e., refined), the closer one can approach SLT’s deterministic ideal. Having said this, it should be emphasized that even partial refinement is far better for diagnostic and/or instructional purposes than simply giving a high level description.

**Structural Analysis (SA): Summary & Benefits:** The method of SA is highly systematic. SA has been partially automated in AutoBuilder with much of the remainder automatable. SA makes it possible to represent knowledge as loosely or as precisely as desired: Advances in hierarchical (AST) representation makes the level of detail arbitrary. Ideally, one represents knowledge at all levels from high-level conceptualization through atomic level. Atomic operations correspond to executables in software and to assumed prerequisites in instruction. In principle, SA can be continued indefinitely as desired. The domain of applicability is automatically specified by AST structures (in SLT content rules).

SA is universally applicable to arbitrarily complex domains. Furthermore SA can be applied incrementally. Domain coverage is indefinitely extendable by adding new kinds of problems. Accordingly, new higher (& lower) order rules must be introduced as needed. SA is a cumulative process, which can builds on prior SA. One can build on existing analyses rather than beginning anew.

The generating power of SA increases monotonically. SLT rules (and higher order SLT rules) identified tend to become simpler as SA continues, while the breadth of coverage and collective generating power goes up qualitatively (e.g., Scandura, 1977).

Rules that generate new rules are not the only higher order rules that play a central role in complex problem solving. Selection rules also play a decisive role where any number of alternative approaches may seem feasible. The ability to decide in such cases is one of the things that separates experts from novices. In production systems and expert systems generally, selection rules correspond to “hard wired” constraints used to resolve conflicts. In SLT, selection rules are fundamentally no different from other rules. In the next section we shall see that lower, and higher order generation and selection rules are all governed by a common set of theoretical constructs. In SLT these constructs govern the way all rules are used, learned and modified.

**Structural (Content) Analysis (SA) Requires Thinking:** Although SA is a systematic and partially automated process, it requires thinking and work. So
why bother? The short answer is that even minimal SA helps. Experience shows that SA adds arbitrary degrees of precision, both increasing efficiency in teaching and learning, and minimizing the need for empirical research. The more detailed a SA, the better. Analysis can be stopped at any point (prior to atomicity). On the other hand, where an SLT rule hierarchy is too detailed, testing and instruction may become redundant and/or inefficient. Achieving a level of atomicity is best.

On the other hand, testing will be less precise (e.g., probabilistic in nature) when analysis is incomplete, thereby requiring multiple test items (cf. Scandura, 2005). In this case, the effectiveness of instruction will be closer to the norm—proportionately less than the guaranteed learning that is theoretically possible. Nonetheless, even informal or preliminary SA is helpful. If done systematically as proposed here, one has the advantage of knowing that SA is cumulative. It can be continued over time without loss as resources allow.

One Final Note: Those familiar with relational networks may ask what higher order SLT rules contribute that can not be done just as well by introducing higher order relationships. The answer lies in the economy of representation. As detailed in Scandura (2005), relational networks (e.g., ALEKS used to represent knowledge of school arithmetic) involve thousands of nodes—versus an estimate of an order of magnitude less. One reason for this large discrepancy is obvious. ALEKS-type representations refer to large integrated relational networks of nodes, involving connections, not only within specific skills (e.g., facts, subtraction and addition algorithm, etc.) but also between such skills. A second reason is subtle, but even more important. The number of nodes in ALEKS-type networks goes up with the square of the number of nodes associated with the various skills (to show relationships between the skills). This dramatic increase arises precisely because ALEKS requires an indefinitely large number of different kinds of relationships.

This is precisely the problem that is solved by SLT rules. Each rule associated with a problem domain, no matter how large or complex, is strictly modular. Relationships between rule “modules” are represented by equally modular higher order rules. Hence, the number of nodes only goes up only linearly with the number of rules. Nothing is lost in the process. Indeed, the introduction of higher order rules actually reduces the number of rules required by eliminating redundant rules.

In addition, rather than arising systematically during the course of SA, relational networks require idiosyncratic evaluation and judgment. Furthermore, we shall see in the next section that learning higher order SLT rules provides a

COGNITIVE THEORY

**Overview and Introduction:** An important distinction in the previous section was made between to be acquired (content) knowledge and what an individual actually knows. Content knowledge is represented as a set of hierarchies of higher and lower order SLT rules. Each hierarchy represents equivalent knowledge at multiple levels of expertise. Higher order SLT rules (which can also be represented in hierarchies) generate new knowledge (e.g., needed to solve novel problems).

In the next section we shall detail how individual knowledge is operationally defined in terms of observable behavior. In this section, we assume we know the relevant rules available to any given individual. This assumption is different from that used in theories based on production systems. The question there is not what individuals know relative to what (content rules) are to be learned, but rather what are ALL of the things (productions) that might be known. This requires identifying all manner of productions, whether they are important to learn or otherwise.

As detailed in the previous section, production system based theories assume restricted control (e.g., “hardwired” chaining) and conflict resolution mechanisms controlling how productions are used. In contrast, SLT assumes a single general-purpose control mechanism that applies universally to all SLT rules, whether of higher or lower order.

As in most information processing theories, learners in SLT are assumed to be goal directed problem solvers. Cognitive theory in SLT details how individual learners, whether human or automated, use and acquire SLT rules. Individual behavior in specific situations is explained, predicted and (indirectly) controlled based on:

1. what SLT rules and higher order rules any given individual knows (and/or is taught and learns) and
2. built-in constraints affecting the way that knowledge is used:
   a) an universal control mechanism (UCM), common to ALL individuals and assumed to control the use of ALL individual SLT rules,
   b) each individual’s fixed processing capacity and
   c) each individual’s characteristic processing speed.
Individuals are assumed to enter each problem-solving situation with a set of lower and higher order SLT rules, each at a specific level of abstraction. UCM controls how every individual uses those SLT rules, subject only to constraints imposed by a fixed (built-in) limitation on each individual’s capacity for processing information and his or her characteristic processing speed.

This section begins by summarizing experimental support for these assumptions. Although much remains to be done, there is substantial scientific support for UCM and some for a fixed processing capacity. The remainder of this section describes the UCM, processing capacity and speed in more detail, particularly the first since UCM will necessarily play a central role in any automated learning system based on SLT that accommodates ill-defined problem solving.

**Experimental Support:** A series of experiments run under laboratory conditions provide strong support for the assumption that individuals are able to solve novel problems if and only if they know requisite lower and higher order rules sufficient for deriving needed solution rules, and UCM is actually available. These experiments were set up in such a way that known rules would be sufficient only if goal switching came built-in as part of the human package (e.g., Scandura, 1971a, 1973, 1974, 1977). Strong empirical support for UCM, involving the use of explicit higher order knowledge in novel problem solving, also has been obtained in field applications (e.g., Ehrenpreis & Scandura, 1974) and in computer simulations (Wulfeck & Scandura, 1977). Indirect support for the central role of UCM in conjunction with higher order automation SLT rules also has been found in in-depth developmental studies with Piagetian conservation tasks (Scandura & Scandura, 1980). Informal observations of two of my grandsons’ behavior further suggest UCM’s availability from infancy (see Scandura, 2003, p. 21-2).

In other experiments, Scandura (1971) and Voorhies & Scandura (1977) found that individuals systematically differ in their processing capacity. Memory loads were calculated for specific procedures (equivalent to SLT rules) for solving a number of simple tasks. Tasks ranged from remembering simple lists, lists supplemented with a fixed constant and simple arithmetic. Performance on these tasks supported the notion that each individual can process only a fixed number of chunks (e.g., 5, 6, 7, 8 or 9) at one time. This number of chunks remained relatively constant for any given individual. Overall, these experiments support the assumption that each individual has a fixed capacity for processing information, irrespective of the task in question—or more accurately, irrespective of the processes an individual uses in solving any given task.

In effect, the results support extension of Miller’s (1956) classic “magic number 7 plus or minus 2” to individuals—not just to group behavior averaged
over individuals as in Miller’s experiments and as adapted in contemporary cognitive load theory (Sweller, 1988).

At this point, support for a characteristic processing speed is limited to common everyday observation. Some people by nature appear to process information quickly—others more slowly and/or deliberately. To ignore this fact of life in psychological theorizing would be like carefully calculating the height of two horses to distinguish the black one from the white one.

**UCM, processing capacity and speed represent fundamental assumptions in SLT:** Collectively, they provide a parsimonious, axiomatic foundation for SLT’s cognitive theory. From this theory one can derive deterministic predications—predications that, when tested under appropriate boundary conditions (Scandura, 1971), must hold for individuals in specific problem-solving situations.

Testing these assumptions empirically has more in common with experimentation in classical physics than experimental psychology. Instead of averaging over groups, one proactively eliminates the effects of confounding variables. In physics, this might involve friction. In SLT it might involve insuring that assumed rules have actually been learned and are available. To make the point, imagine that we have programmed a computer with a set of SLT rules and that we give the computer a task to be performed. What will happen? The answer is nothing—at least not until we tell the computer what to do. Should rules be tested one by one? Should they apply whenever an input matches? Should they be chained, should higher order rules be applied to other rules? And if so, when?

Testing for the presence of UCM, for example, involves doing the same thing with human beings. Assuming UCM is universally available, we build in the assumed rules and present the problem. Then, we see what happens. Under idealized circumstances, where the learner understands the problem and we really know what rules the learner already knows (and does not know), SLT should be able to predict precisely whether that individual will or will not be able to solve the problem.

Determining whether or not each individual has a characteristic processing capacity and/or speed requires similar initial conditions. Clearly, any individual’s capacity for processing information will depend on the processes actually being used. It is well known, for example, that one can process more information by chunking information. To determine what an individual’s capacity is, it is essential to know exactly processes the learner is using—or equivalently, what chunks are being formed and/or used in the process.

The above experiments were conducted with these principles in mind. These
assumptions play a fundamental role in SLT’s cognitive theory. Accordingly, independent research to confirm and/or extend these findings is strongly encouraged. There are, for example, no directly related experiments that I know of pertaining to establishing a characteristic processing speed for individuals. Furthermore, while it is fairly clear how and when new information enters a fixed capacity processor, the order in which information drops out is not. Most likely, it is not something like “first in, first out”, or “last in, last out”. Rather, it is more likely to be something like “most distantly used”. Deriving and testing implications of these assumptions is likely to be even more revealing. We give an important example of such research in the concluding section.

**Automating UCM, Processing Capacity and Speed:** The general architecture of such a system is shown in Fig. 7. This schematic shows that there is little difference in SLT between short-term (working) memory and long-term memory. Given a problem, UCM controls the use of all SLT rules in working memory. The UCM, together with available SLT rules, not only determines what new information (e.g., rules) may be derived but also what can be retrieved from long-term memory at any given point in time. There is no essential difference, other perhaps than efficiency of the higher order SLT rules used, between retrieving previously learned rules and deriving new ones.

**Problem Solver / Learner Architecture**

![Diagram](image.png)

**FIGURE 7**
Schematic of the problem solver architecture and its interactions with an external agent.
In an important sense the Universal Control Mechanism (UCM) represents a least common denominator with minimal assumptions about the way available SLT rules are used and new ones are generated (i.e., learned). Initially, UCM was described in terms of simple goal switching: Given a problem, the learner checks for rules whose domains and ranges match. If such a rule is found, the rule is applied. If not, the goal switches recursively to higher-level goals for deriving a rule that does apply. Once such a rule is found and applied, the result is added to the set of available rules and the process reverts to the previous level. This time, a matching rule is found, applied and the problem solved.

Initially, goal switching was assumed to take place only when no matching rule was found (Scandura, 1971). A couple years later it was extended to account for data involving selection from among two or more matching rules (Scandura, 1973, chapter 8). Rule selection in this context corresponds to conflict resolution in expert systems and motivation in psychology. The higher order rules used in various experiments (see above) ranged from the equivalent of means-ends analysis and chaining to learning by example (cases). Variations on each such mechanism have been widely used in automated (e.g., expert) systems. Invariably, however, these mechanisms (or some combination thereof) are “hard-wired”.

Although conceptually simple and empirically powerful, it became painfully clear when we tried to automate goal switching that it is very difficult to implement in a manner that is completely independent of the higher order rules involved.

A simulation program developed by Wulfeck and Scandura (1977), for example, successfully demonstrated empirically progressive learning over time. We found it impossible, however, to completely separate higher order rules from control. In an attempt to explain transitions more generally, Scandura (1981) included a footnote, whose importance was only partly understood at that time. Rather than viewing control in terms of goal switching, view it from the standpoint of where to look in matching rules. If no rule matches look for matches in the ranges of the available rules.

In retrospect, this observation provided the key to solution. The idea was later detailed and reduced to practice in Scandura (U.S. Patent 6,275,976, 2001, esp. Figs. 26 & 26A). An overview of this method follows:

- 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)

To see how this works suppose that a person is given, taught or otherwise knows how to convert yards into feet and feet into inches. The person is presented with the problem of converting a given number of yards into inches. Some (most educated adults) will be able to do so. But, others will not. Why not? The difference lies in whether the problem solver knows how to combine known rules of the form A → B, B → C — i.e., where the output of the first matches the input of the second. To solve the problem, the learner must “chain” the rules; that is, perform them one after the other. As Scandura (e.g., 1971, 1973, 1974) demonstrated experimentally, higher order rules such as this can be identified, taught and learned independently of the problems themselves. Young children who failed to solve such problems initially, except for isolatable errors of measurement, were uniformly successful after learning how to compose rules in which the output of one was input to a second. Both the inputs and outputs involved in the higher order training were themselves rules. Furthermore, transfer was tested on novel problems the learners had never seen before.

Exactly the same results were found in a series of experiments involving generalization from examples (Scandura, 1974). It’s interesting to note that Case Based Reasoning is based on the same principle—the main difference being that CBR simply assumes the availability of higher order generalization rules. This research suggests that more attention should be given to identifying and teaching such higher order rules.

These ideas are illustrated in more detail in Examples 3 and 4 below:

EXAMPLE 3

<table>
<thead>
<tr>
<th>Givens</th>
<th>Goal</th>
<th>Givens</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>rug {messy&amp;dirty}</td>
<td>? rug {clean}</td>
<td>OR</td>
<td>messy&amp;dirty—?clean</td>
</tr>
</tbody>
</table>

That is, a rug that is messy and dirty is to be made clean. The former representation is how problems and rules (below) are written in AuthorIT’s HLD language. The latter is an easier to read short cut notation.
Rule Set

Lower Order SLT Rules

(2) vacuum (dirty; clean) OR dirty→clean
    pickup (messy&dirty; dirty) OR messy&dirty→dirty

Higher Order SLT Rules

(3) chain [SLT-rule1 (A; B), SLT-rule2 (B; C): SLT-rule (A; C)]
    OR more simply as
    A→B, B→C → A→C,

The range of this higher order rule is SLT-rule (A; C) OR A→C

In this example, no lower order SLT rule (alone) in the Rule Set matches the problem. Hence, control seeks (higher order) rules whose range matches the problem goal.

The range of the higher order rule (3) is A→C, which matches—that is, has the same structure as messy&dirty—?clean

According to UCM, what happens in this case is the following:

1. The Range Structure (A→C) of the higher order rule matches the original Problem Structure (messy&dirty—?clean).
2. UCM control seeks to match the higher order rule’s domain against the set of available SLT rules. In this case, dirty→clean matches B→C and messy&dirty→dirty matches A→B.
3. The domain of the higher order rule (3), that is A→B, B→C, is satisfied by lower order SLT rules messy&dirty→dirty and dirty→clean in step 2.
4. The higher order SLT (chaining) Rule (3) is then applied to messy&dirty→dirty and dirty→clean.
5. The newly generated solution rule messy&dirty→(dirty→)clean is added to set of available rules.
6. Control checks the original problem against the rule set enhanced with messy&dirty→clean.
7. Control reverts to previous level where the newly generated messy&dirty→clean rule matches. Hence,
8. This rule is applied and solves original problem

EXAMPLE 4

<table>
<thead>
<tr>
<th>Givens</th>
<th>Goal</th>
<th>Givens</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) bed {unmade}</td>
<td>? bed{made}</td>
<td>OR</td>
<td>unmade ? made</td>
</tr>
<tr>
<td>rug {dirty}</td>
<td>? rug {clean}</td>
<td>OR</td>
<td>dirty ?clean</td>
</tr>
</tbody>
</table>
A bedroom is to be made presentable. This requires that the bed is made and the carpet is clean.

**Rule Set**

**Lower Order SLT Rules**

(2)  
make (bed) OR unmade→made  
vacuum (rug) OR dirty→clean

**Higher Order SLT Rules**

(3)  
do_in_parallel (:A1→B1, A2→B2; ) OR  
A1→B1, A2→B2 → A1→B1  
A2→B2

This higher order rule generates the two domain rules that are to be executed in parallel (order is not assumed to be important). As in Example 3, no lower order SLT rule (alone) in the **Rule Set** matches the problem. Hence, control seeks (higher order) rules whose range matches the problem goal. The range of the higher order rule (3) is

\[ A1 \rightarrow B1 \]
\[ A2 \rightarrow B2 \]

This pair of rules has the same structure as the problem goal:

\[ \text{unmade ? made} \]
\[ \text{dirty ? clean} \]

Hence, according to UCM:

1. The **Range Structure** of higher order rule (3) matches Problem Structure (1).
2. UCM control then seeks to match the domain of higher order rule (3) against the set of available SLT rules (2).
3. The domain of this higher order rule (3) is satisfied by the two lower order SLT rules shown in (2).
4. The higher order SLT (3) is applied to the SLT rules (2).
5. The newly generated solution rule
   \[ \text{unmade} \rightarrow \text{made} \]
   \[ \text{dirty} \rightarrow \text{clean} \]
   is added to set of available rules.
6. Control checks the original problem against the rule set enhanced with the rule (pair) in step 5. It matches. Hence,
7. Control reverts to previous level where newly generated rule in step 5 matches, is applied and solves original problem

Exactly the same sequence of steps applies irrespective of the higher and lower order rules involved. Space limitations preclude further analysis herein. Nonetheless, the motivated reader is encouraged to trace operation of the UCM with each type of higher order rule in the section on Structural Analysis.

**Transitions:** It is particularly interesting to trace the process by which knowledge progresses from naïve through neophyte to expert. As above, we’ve seen how higher order SLT rules result in the acquisition of new SLT rules. The latter make it possible to solve what we shall call ill-defined problems, which are beyond the scope of immediately available knowledge. We have also discussed, albeit more briefly, the mastery process whereby higher order automation rules convert procedural knowledge to structural.

This process of automation plays a crucial role in transitioning from one level of knowledge to the next. We have already detailed how UCM accounts for local transitions. Mastery of reading and writing numerals (e.g., assembling line segments to write “5”, “7”, etc.) is prerequisite to learning arithmetic algorithms. It is difficult, indeed likely impossible to learn arithmetic algorithms until a child has mastered reading and writing numerals. In effect, the latter is an essential prerequisite.

What may not be clear from the above is that we have come full cycle, a cycle that can be repeated continuously throughout human (potentially as well as in automated) learning and development. Transitions from naïve to neophyte are the result of higher order rules generating new rules by operating on lower order rules. Transitions from neophyte to expert are the result of higher order automation rules converting procedural knowledge to more efficient and direct structural knowledge.

Once knowledge has been automated (i.e., mastered) it provides the foundation necessary for the next level of learning. It is essentially impossible to define a problem until the goal of that problem has been automated—i.e., is structural in nature. Put differently, the learner must be able to match problem goals with solution outputs. This applies whether the cycle is local in nature, as in learning to write numerals, or more global. In each case, one level of learning provides the foundation for the next. This issue has been the source of much controversy in developmental psychology (e.g., Smith, 1992).
While the SLT account is considerably more precise, this cycle shares much in common with Piaget’s well-known developmental levels. The importance of automation in this context first became clear in a series of studies conducted by my wife and I (Scandura & Scandura, 1980). It was simply impossible to define conservation tasks in an operational matter until critical prerequisites have been mastered. For example, we found that young children cannot conserve number (as defined by Piaget) in any reasonable way until they have mastered one-to-one matching. Exactly the same principle held with liquid, weight and other types of conservation. More interestingly, we found there was no one level of conservation, but rather various kinds of conservation. A child might learn to conserve number, for example, but not weight or vice versa. In short, we were able to experimentally manipulate what Piaget called “horizontal decalage”.

**Summary:** Advances in Structural Analysis (SA) and Universal Control Mechanism (UCM) provide a general and executable foundation for supporting ill-defined problem solving and intelligent behavior generally in arbitrarily complex domains.

1. **Structural Analysis (SA)** provides a systematic method for identifying needed higher as well as lower order knowledge.
2. **Universal Control Mechanism (UCM)** provides a general mechanism for controlling interactions among arbitrary sets of higher and lower order knowledge (U.S. Patent 6,275,976, Scandura, 2001, 2003).

SA underlines the importance of—indeed requires—separating higher order knowledge from control. Higher order knowledge (e.g., chaining, generalization, analogy, conflict resolution/rule selection, etc.) is both crucial in all intelligent (e.g. novel problem-solving) behavior and fundamentally context dependent. Accommodating the indefinite number of variations possible literally demands a “hard wired” control mechanism—one that is universally applicable.

Implementing UCM as a working program will obviously require more detail than can be described here. For direct reference, Flexform designs for UCM taken from Figures 26 and 26A in Scandura (2001) are included below as Figs. 8A & 8B. Although the terms used may differ slightly, fundamental concepts and ideas remain essentially the same.

**Fixed Processing Capacity:** Psychological research traditionally involves subjecting various groups of human subjects to different experimental conditions.
The effects of uncontrolled variables are eliminated by randomly assigning subjects to various treatments. This approach can easily camouflage mechanisms used by individual subjects. It is possible to explain incrementally observed improvements in learning via diametrically opposed assumptions about how individuals actually learn. In the late 1960s, for example, experimental psychology went through a long period of research focused on understanding the mechanisms of learning. This led to the development of theories such as the Universal Control Mechanism (UCM) and the Flexible Learning Framework (FLF), which provided a framework for understanding how learners interact with educational materials.

FIGURE 8A
Figure 26 in Scandura (2001) is a Flexform detailing higher levels in Universal Control Mechanism (UCM).
period of debate over whether (individual) learning took place on an incremental basis as so-called functionalists maintained (e.g., Melton 1963) or whether learning was all-or-none (e.g., Peterson & Peterson, 1959; Greeno & Scandura, 1967).

This example clearly demonstrates how difficult it can be in studies involving groups of learners to distinguish even diametrically opposed assumptions about individual learning mechanisms. Definitive research requires carefully controlled
studies with individuals (cf. my comments on Galileo’s thought experiments at the Leaning Tower of Pisa and their impact on early physics).

The task is even more demanding with regard to processing capacity. Miller’s (1956) classic results clearly were the result of averaging. Not only must one distinguish differences between individuals but also between different procedures used by individuals on the same tasks. In our original research on this subject (Scandura, 1971, 1973; Voorhies & Scandura, 1977), the memory load imposed by each process-task combination was determined based on the number of chunks (digits) a subject must maintain in memory at each stage of processing to complete the task. Memory load was determined based on detailed analysis of the processes to be used.

Individual Ss were next subjected to intensive pre-training; they were carefully rehearsed to insures that they actually used those processes. A metronome, for example, was used to help minimize the possibility of uncontrolled (e.g., auditory) chunking. The tasks used ranged from simple lists of digits, with and without “1” as a fixed starting digit to simple addition tasks similar to those used by Suppes & Groen (1968; see Scandura, 1973, 1977).

More generally, the duality between structural and procedural knowledge in SLT rules provides a natural way to compute memory load that can be made arbitrarily precise. The memory load associated with any SLT rule used by any particular individual depends on the number of structures that must be kept in memory at any one time. Expert knowledge of subtraction, for example, might impose a memory load of only one where the answer to a problem like (100 - 50 =?) is known immediately—i.e., when the problem is encoded (i.e., chunked) as a whole, and solved in a single step. Neophyte knowledge of column subtraction, on the other hand, requires reference to individual digits and essential comparisons (7 < 9) at various points in carrying out the solution procedure.

This (last) example also makes clear the importance of what I have called memory free (cognitive) theorizing (Scandura, 1971). The memory load imposed on an individual when carrying out any particular procedure (e.g., the subtraction algorithm) with paper and pencil is quite different than doing everything “in one’s head”. Sodoko puzzles (filling in 9 squares with digits so that each square, row and column uses each digit exactly once) provide a clear example. Solving such puzzles without pencil and paper is beyond the ability of most. While a “photographic memory” might help, the sheer number of different but overlapping “mental photographs” suggests that even this would be inadequate.

The difficulties involved are analogous to those faced by experimental physicists, some of whom have devoted careers to testing critical assumptions in relativity and quantum theory. If testing such assumptions is so difficult, then why
would anyone care—especially those of us concerned with using technology to improve education?

The answer lies in the generality and potential precision of such results. Interest in the role of memory load in instruction has increased in recent years—with the basic assumption essentially being that more complex tasks impose a greater memory load during learning than simpler tasks. Some (e.g., van Merrienboer, 2006) have even argued that transfer is increased by reducing the amount of instruction, thereby increasing memory load. Whether or not this is true in some general sense (averaged over individuals), any such result from the present perspective is largely incidental. Perhaps reducing instruction or increasing memory load forces the learner to do more on one’s own (i.e., assuming he or she is successful). The essential factor, in each case, is what is actually being learned. SLT related research demonstrates rather conclusively that one can explicitly teach transfer (e.g., by identifying and teaching needed higher order rules).

More important (than averaged results), some people are able to handle more complex tasks than others. Being able to pinpoint an individual’s processing capacity has implications that go far beyond the particular task in question. The memory load imposed by any particular task (actually solution procedure) can be determined analytically—without collecting experimental data (cf. Scandura, 1973, 1977). While definitive experiments remain to be done, the structures operated on in any particular SLT rule provide an explicit basis for computing memory load. Simply count the number of such structures involved at each state of the solution procedure, and take the maximum.

When more than one SLT rule is involved, as when higher order rules are needed, the simplest approach would be to determine maximum number of structures being operated on—although this can only be confirmed experimentally. UCM does dictate a particular sequence of activities. Hence, it seems fairly clear which and when new structures are generated, and hence added to working memory at each point in time. The opposite is not so clear: Which elements are to be dropped out when processing capacity is exceeded? The answer to this question obviously can only be determined through highly rigorous experimentation with individuals.

My guess is that it will not be something like “first in, first out” or the opposite. One reasonable hypothesis is that structures used most distantly in the past would be the first to go. Whether the results support determinism, or otherwise—perhaps nondeterminism—remains to be seen.

In any case, determining an individual’s processing capacity opens a whole new set of possibilities. It only needs to be done once. Memory load can be determined in a purely analytic manner, thereby providing a sound foundation for broad ranging predictions. Potential applications range from selecting individuals able to process
requisite, large amounts of information to deciding how much information an individual can handle during learning. Such information could be extremely useful in adapting instruction (or predicating the behavior) of individuals.

While deterministic results are theoretically possible, application of these ideas does not require this degree of precision. Even rough approximations would make it possible to address individual differences more effectively than is possible in traditional research based on groups.

**Characteristic Processing Speed:** Common every day observation strongly suggests that individuals differ in the rate at which they process information. Nonetheless, as mentioned earlier, there is little or no formal experimental research on the subject. Separating the effects of over learning (e.g., chunking) from processing speed will require experimental methods in the same spirit as those used in identifying processing capacity. The rate at which any given individual performs any given task will at minimum depend on the mix of structural and procedural knowledge. Clearly, operating at higher levels of abstraction/mastery almost certainly will be faster (for any individual) than at lower levels (i.e. carrying out the same task procedurally). Comparison of individual processing speeds must control for such differences. Finer grained questions involve encoding (i.e., of structures) and cognitive (i.e., procedural) speeds.

Despite experimental complications, demonstrating that individuals process what they know at characteristic, determinable rates could open an entire range of new possibilities. The general (as opposed to task specific nature) of these characteristics would enable selecting people based on their potential for particular jobs—individually of whatever prior experience they may have had. The implications of such demonstration for testing would be profound, with enormous implications including selection based on potential rather than prior opportunity.

**Summary and Discussion:** UCM and associated processing capacity and speed have considerable potential whose implications have only begun to be explored. Considerable experimental support for UCM already exists in the literature (Scandura, 1971, 1973, 1974, 1977). The ability to add, modify and/or delete higher order rules at will without modification to (hardwired) control has considerable potential for automation and should be explored.

Observational support further suggests that UCM may come built-in from the earliest ages. My nine-month-old grandson, Sam, was able to get up on his hands and rotate his body, but every time he tried to crawl he moved backward. This proved very frustrating as he tried to move toward a favorite toy. Then, instantly he rotated 180° crawled backward, moving toward the toy. He gained obvious
satisfaction by again rotating 180°. My interest here was not in Sam’s use of a higher order chaining rule, or my other grandson, Zach’s use of higher order analogies. Far more satisfying was knowing that the only way that Sam’s or Zach’s higher and lower order knowledge could be used was to have the built-in equivalent of UCM. Clearly, UCM also has important potential implications for automated intelligent systems.

Direct experimental support also exists for the assumption that each individual has a fixed processing capacity (e.g., Scandura, 1971, 1973; Voorhies & Scandura in Scandura, 1977). These results, of course, are tentative and should, along with processing speed be subjected to rigorous experimental test.

The need for more fundamental research seems clear, and I encourage psychologists and others trained in experimental methods to accept the challenge. Nonetheless, enough already is known to warrant broad application in educational research and particularly in the development of new and better kinds of instructional systems (cf. Scandura, 2005).

ASSESSING BEHAVIOR POTENTIAL

As emphasized above there is an essential difference between SLT Content Rules and what any given individual knows. SLT content rules represent what is to be learned—the knowledge necessary for success in the given domain. Content knowledge includes both higher and lower order SLT rules, each of which can be represented at multiple levels of abstraction, representing various degrees of expertise. In the last section we detailed fundamental mechanisms and constraints that determine how individuals use available knowledge.

In this section, we address a remaining critical question: Can individual knowledge be operationally defined in terms of observable behavior? According to SLT’s cognitive theory, this requires determining what any given individual knows relative to each of the higher and lower order content SLT rules (associated with some given domain).

We shall see below that individual knowledge with respect to any given content rule can be represented relative to directed graphs, or more generally, as overlays on SLT rule hierarchies. The former represent individual behavior potential at a specific level of expertise (abstraction)—equivalently, relative to predetermined assumptions as to available prerequisite knowledge. The latter is more general, and represents individual behavior potential, with respect to all levels of expertise.

We begin with a short history based on the use of directed graphs (flow charts) for this purpose and show how SLT rule hierarchies offer a more complete solution.
**Representing Individual Knowledge:** Individual knowledge in SLT was originally defined in terms of paths through **directed graphs**. The following illustration is adapted from Scandura (1971). Figure 9 shows both the total graph and sub-graphs (or paths) through a solution rule for constructing the next numeral in base 3. The arcs correspond to operations, which are assumed to act in **atomic** fashion. That is, success on any one instance of such an operation is tantamount to success on any other, and similarly for failure. The nodes correspond to decisions made in carrying out the algorithm on particular test items.

The **subgraphs** at the bottom of Figure 9 correspond to the four possible paths through this directed graph, each of which is used in solving a different type of problem. Since the constituent rules are all assumed to be atomic, it follows that each of these paths also acts in atomic fashion. Hence, one test instance for each path is sufficient to determine the behavior potential of a given learner. In this example, the base-three **stimulus → response** numerals, \(101 \rightarrow 102, 2 \rightarrow 10, 112 \rightarrow 120, \) and \(222 \rightarrow 1000\), correspond respectively to the four possible paths. Accordingly, the behavior potential of any given subject on this class of tasks can be uniquely specified by performance on just these four test instances—as long as the atomic assumption is valid.

An extensive body of research demonstrates that an individual’s behavior potential can be determined and predicted with near 100% accuracy when content knowledge is sufficiently refined so that individual nodes are atomic (meaning that they are so simple that every learner in the intended population will either know each perfectly or not at all). In practice, atomicity is often hard to achieve. Nonetheless, this level of precision has been attained in experiments run under laboratory conditions, and represents an ideal that can be approached to an arbitrary degree—even with higher order knowledge (cf. Scandura, 1971, 1973, 1977; Durnin & Scandura, 1973). Predication levels have exceeded 70% even in real world applications (e.g., Ehrenpreis & Scandura, 1974; Scandura, 1977).

While they clearly define what parts of a given rule an individual does and does not know, **directed graphs fail to address levels of expertise**. Latency or rapidity of response is commonly used to differentiate expertise (or atomicity) in psychological experiments. **Latency provides an important and operational alternative**, but it misses one essential captured by AST-based SLT Rules—the **complementary relationship between structural (declarative) and procedural knowledge**. Separating the two requires taking into account both the structural, largely unconscious encoding and the cognitive, primarily conscious processing required in solving any given task. SLT rule hierarchies provide the
representational richness necessary for this purpose. This richness is important because it makes such differences explicit.

A second major limitation of directed graphs is that they do not provide an automated way to construct problems associated with any given path. That is, given a path (through a directed graph) there is no way to construct automatically a problem that will exercise precisely that path—something that is essential in diagnosing what a learner does and does not know. In long division, for example, there is no easy way to know ahead of time (without carrying out the procedure) whether or not a trial divisor may have to be reduced. The fact that one can always find such a problem by trial and error does not solve the problem of how to construct such problems automatically.

Representing knowledge in terms of SLT rule hierarchies solves this problem as well. SLT rule hierarchies are fully interpretable on a computer, and by definition executable. With the help of a simple, general-purpose interpreter, SLT rule hierarchies can be executed automatically. Each step in such an execution corresponds to a unique node in the hierarchy and produces a unique, definable state corresponding precisely to a partially solved problem.
Collectively, these nodes define all possible sub-problems at multiple levels of abstraction. Given a problem, each node in an SLT rule hierarchy uniquely defines a unique sub-problem of that problem. The number and kinds of nodes exercised in solving any given problem depends on the nature of that problem. Simpler problem types (templates) exercise relatively few nodes. More complex ones exercise more.

Put differently, each problem template corresponds to one or more root nodes in the corresponding SLT rule hierarchy. Solving such a problem (template) by executing the SLT rule hierarchy will exercise all or portions of the nodes below that root node. The sub-problems automatically generated in the course of solution provide a sound, efficient basis for diagnosis (patent pending).

This discovery dramatically reduces the number of problem types that need to be constructed. While one can construct as many problems as desired, one can reduce the number of different problems required to provide full coverage by choosing problems wisely. Each step in solving any given problem (template) effectively defines a whole set of sub-problems. These

<table>
<thead>
<tr>
<th>Adding “ing”</th>
<th>Column Subtraction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem Templates</strong></td>
<td></td>
</tr>
<tr>
<td>xxx&lt;silent e&gt;</td>
<td>5 3 9</td>
</tr>
<tr>
<td>xxx&lt;consonant&gt;</td>
<td>- 3 6 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Short Description of each SLT rule hierarchy</strong></th>
<th><strong>Drop silent e, add “ing”</strong></th>
<th><strong>Subtraction algorithm w/ simple borrow</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Diagnostic Sub-Problems</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>date → dating</td>
<td>5 3 9 → 8 3 9</td>
</tr>
<tr>
<td></td>
<td>- 3 6 2</td>
</tr>
<tr>
<td>run → running</td>
<td>5 3 9 → 5 3 9</td>
</tr>
<tr>
<td></td>
<td>- 3 6 2</td>
</tr>
</tbody>
</table>

sub-problems correspond to unique segments of different solution paths in directed graphs.

As shown above, each problem template serves as a diagnostic instrument. Problem templates and diagnostic sub-problems corresponding to these templates
are shown below. Applying (executing) the associated SLT rule hierarchy to a template generates a set of (one or more) diagnostic sub-problems.

Clearly, neither of these templates is complete. An ideal prototype (problem template) will exercise as many nodes as possible. A complete template for the borrowing node in column subtraction, for example, would include both columns that involve borrowing across zeros and those that do not. In column subtraction one can construct a prototype that exercises every node below the top-level node. Such a problem template would involving no borrowing as well as both types of borrowing.

Looked at from a different perspective, nodes in procedural ASTs uniquely define a set of sub-problems.

(NOTE: This perspective is equivalent to our original approach to assessing behavior potential based on directed graphs (or Flow Charts). Terminal nodes in an SLT rule hierarchy effectively define a directed graph representing the lowest, most detailed level of representation of what is to be learned.)

All sub-problems in this set are necessarily sub-problems of the sub-problems defined by the given node. Any one would serve equally well for purposes of diagnosis. In Figure 3, for example, the highlighted node defines a subclass of subtraction problems involving borrowing.

To summarize, diagnosis or assessing behavior potential involves determining known and unknown parts of SLT rule hierarchies. Given any problem type, an SLT rule hierarchy automatically defines a set of diagnostic sub-problems for all nodes exercised in solving problems of that type. Different problem types exercise different sets of nodes in the associated SLT rule hierarchy. Problem types associated with higher-level nodes generally define larger numbers and/or varieties of sub-problems. Accordingly, they provide the most diagnostic power with the least manual effort (in terms of problem construction). An ideal problem type, when such exists (e.g., as in column subtraction), will exercise all nodes in any given SLT rule hierarchy.

Given sufficient precision (i.e., atomic refinement) research demonstrates that a single test item is sufficient to determine whether or not a learner knows the corresponding node (or equivalent sub-path of a path in a directed graph). A learner’s current state of knowledge with respect to any node in a SLT rule hierarchy may be represented by assigning “known” (+), “unknown” (-) or “undetermined” (?) to that node. This assignment will be deterministic in cases where Structural Analysis (SA) reaches the point of atomicity. As we shall see, when analyses are incomplete, probabilities or success on multiple test items may be required (in order to determine mastery).
Distinguishing Knowledge Representations: The reader at this point may be wondering whether our cohesive and otherwise appealing account is built on a house of cards. In particular, there is any number of different ways of solving problems in any given class. While this is true theoretically, instruction is invariably limited to those few solution methods endorsed by subject matter experts (SMEs) that have withstood the test of time.

In column subtraction, for example, there are only two methods widely taught in schools: borrowing (as in our example) and equal additions. I personally learned equal additions in school and it is still taught in Europe and other parts of the world—most likely because it avoids complications due to borrowing across zeros. (Borrowing later became popular in the USA because it is arguably easier to rationalize in terms of place value and concrete realizations thereof.)

The question here is how to distinguish between alternative accounts of the same behavior? As illustrated in Durnin and Scandura (1973), one way to determine “best fit” is to identify problems corresponding to the intersection of the equivalence classes of problems associated with paths through alternative solution rules. Test/retest reliabilities on test items in the intersection classes defined by borrowing and equal additions were in the 70-91% plus range far exceeding what one normally expects in testing.

For example, “equal additions” does not distinguish between (i.e., require different solution paths in) the two problems below. “Borrowing” does!

\[ \begin{array}{c}
7 \quad 16 \\
-4 \quad 3 \quad 8 \\
3 \quad 8 \\
\end{array} \quad \begin{array}{c}
5 \quad 10 \quad 15 \\
-3 \quad 2 \quad 43 \quad 7 \\
2 \quad 6 \quad 8 \\
\end{array} \quad \begin{array}{c}
6 \quad 7 \quad 16 \\
-3 \quad 8 \\
3 \quad 8 \\
\end{array} \quad \begin{array}{c}
4 \quad 5 \quad 9 \quad 10 \quad 15 \\
-2 \quad 3 \quad 7 \\
2 \quad 6 \quad 8 \\
\end{array} \]

(Note: If a learner knows more than one solution rule (e.g., equal additions & borrowing), then predicating (as opposed to just assessing) what the learner will do depends on the operative higher order selection rules.)

Distinguishing Levels of Expertise: We have already discussed issues involved in distinguishing atomicity level in (i.e., what a learner knows relative to given) SLT rule hierarchies. Higher levels in such a hierarchy correspond to higher levels of expertise—corresponding to higher-level operations on more complex structures. One tests higher levels of expertise by testing on problems (or sub-problems) as wholes (corresponding directly to the structures involved), ignoring lower levels of procedural detail. Put differently, higher-level operations correspond to (essentially are) atomic rules operating on relatively complex structures—equivalent to
declarative knowledge. The result is both faster execution and the irrelevancy, or inability to distinguish the actual internal procedures used.

Given a subtraction problem, for example, a subtraction expert (or idiot savant) quickly writes the difference. The neophyte needs more time to go through the steps.

Working at the top level of abstraction, the expert solves the problem in his or her head. The procedure acts on subtraction problems as a whole. Interim steps, whatever they may be, are simply ignored.

The neophyte, by way of contrast, goes through steps at lower levels in the corresponding lower-level SLT rule hierarchy.

As detailed above learner models can be defined in two distinct ways: (1) in terms of paths through a directed graph in which individual arcs and nodes correspond to terminals in an SLT rule hierarchy and (2) as an overlay on an SLT rule hierarchy. In the latter case, nodes at the highest levels in the SLT rule hierarchy correspond to arcs (and nodes) in paths associated with higher-level SLT rules—in which the operations (i.e., arcs) operate on more complex structures.

These higher-level SLT rules correspond to higher levels of expertise. The ability to solve the same kinds of problems as an expert does not necessarily imply expertise. For example, just because a person can perform the subtraction algorithm perfectly does not mean that the person can solve subtraction problems as efficiently as an expert. Accurate characterization of expertise requires operations on more complex structures. Rather than individual steps in an SLT rule operating on individual digits, for example, those steps in the case of an expert might operate on entire columns (whether or not they involve regrouping) or even entire subtraction problems.

Distinguishing expertise when the behavior is equivalent presents subtle issues. Psychologists have typically used latency, or the speed at which a correct response is given, to distinguish levels of expertise. Psychologists have not, however, typically equated latency with the aforementioned distinction between procedural and declarative knowledge. Levels of expertise also can be distinguished via the solution procedures used. Whereas the neophyte, for example, might use regrouping (or equal editions) in column subtraction, the expert is more likely to use shortcuts involving multiple columns. Individual steps the expert uses may be immaterial as long as one gets the correct answer. Intermediate steps are ignored.

\[
\begin{array}{c}
\begin{array}{c}
4 \ 3 \\
-2 \ 7 \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
4 \ 3 \\
-2 \ 7 \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
1 \ 6 \\
\end{array}
\end{array}
\]
In short, neophyte versus expert knowledge can be distinguished by measuring latency and/or by restricting attention only to essential responses—ignoring intermediate steps.

Both techniques are used, for example, in TutorIT (Scandura, 2005). Authors can define automation nodes and assign latencies that must be met as a condition for mastery. Authors can also specify nodes, corresponding to intermediate steps, where responses are to be ignored. In the latter case, the learner is allowed to simply produce the correct answer, ignoring intermediate steps.

Assessing Higher Order Knowledge: Assessing Behavior Potential with respect to Higher Order SLT rules is essentially identical. The only difference is that the Domain and Range elements in the SLT rule’s data structure, and correspondingly the Givens and/or Goals of the problems used for diagnosis, must include other SLT rules. Although conceptually identical, however, there are important differences in application that require special attention. For example, consider the simple higher order SLT chaining rule below.

\[ A \rightarrow B, B \rightarrow C \rightarrow A \rightarrow C \]

The inputs to this higher order SLT rule are themselves SLT rules and must be presented as such during testing. Special precautions were made in this regard in our earlier cited research on the Universal Control Mechanism (UCM) (e.g., Scandura, 1971, 1973, 1974, 1977).

(Note: Although the above schematics look identical to condition-action productions, productions are very different from SLT rules. The data structures in SLT rules represent abstract syntax trees rather than conditions (relationships). Similarly, the procedures in SLT rules include but are rarely limited to simple operations.)

As emphasized above, knowing a SLT rule includes both its procedure and the data structure on which it operates. Domains play an especially important role in higher order knowledge. Theoretically, for example, higher order rules corresponding to forward and backward chaining may be applied universally. This is also true of SLT rules representing inference, generalization, abstraction, selection, automation, etc.

In practice, however, essentially all SLT rules, both higher and lower order, have restricted domains of applicability. Overgeneralization of SLT rules is common in the real world. Accordingly, it is not surprising that SLT rules are often misapplied (i.e., generate responses that do not satisfy problem goals). Simply labeling SLT rules (as above) ignores the potential
complexity of both the data structures and procedures. **AST structures have the important advantage of making it possible to define domains of applicability with arbitrary degrees of precision**—with systematic analysis effectively reducing the need for and/or increasing the precision of experimental data.

As seen in the previous section, the UCM makes it possible to predict behavior on complex problems whose solution requires both Higher and Lower Order SLT rules. In complex domains, therefore, **it is sufficient to assess behavior potential with respect to lower and higher order SLT rules individually.**

**Summary:** As detailed above, knowledge attributed to individuals is measured relative to SLT content rules (i.e., operationally defined in terms the learner’s behavior). Individual knowledge is represented by a learner model consisting of an overlay on nodes in the given set of SLT rule hierarchies. Individual behavior potential is represented by “marking” nodes in the associated SLT rule hierarchies as known, unknown or yet to be determined.

Each node in an SLT rule hierarchy defines the kinds of problems an individual can solve. The nodes collectively define both the individual’s behavior potential and level of expertise. The individual’s behavior potential (the behavior the individual is capable of) at any particular level of abstraction can be represented as a sub-graph of a directed graph—for example, a directed graph representing terminal operations and decisions in a SLT rule hierarchy.

**Discussion:** Production rules are variously assumed to reflect some degree of biological reality. Knowing relevant productions (solicited from SMEs), along with fixed control and conflict resolution mechanisms (and “probabilistic” memory constraints) is assumed sufficient for determining both human behavior and the kind of instruction needed.

**SLT rules attributed to an individual may or may not have a direct**


counterpart in brain physiology. SLT rules focus on and derive from observable behavior. In that sense, SLT rules serve a different purpose than production rules (cf. in Anderson’s ACT theories, 1988).

Knowledge available to individuals in SLT is measured relative to (sets of) SLT rule hierarchies. Whether or not learners actually use operationally defined individual SLT rules, or equivalent productions as in model tracing, is essentially irrelevant. One cannot tell the difference by looking at the learner’s behavior. This is true whether or not one is interested solely in what should be learned or so-called “buggy” knowledge as well. The latter case would be accommodated in SLT by introducing higher order selection rules (see above). “Bug-ridden” learners would have to learn to select correct as opposed to inappropriate rules.

From an instructional point of view, it doesn’t make any difference whether these constructs have any corresponding reality in the brain, or whether they are simply convenient theoretical constructs? As instructional scientists we don’t have license to look inside. We can only infer what any given learner knows from his or her observable behavior. It makes no difference what is actually inside. We cannot program learners.

We can only assess what any given individual knows relative to targeted knowledge, or more accurately observable behavior made possible by that knowledge. In instruction, what is of interest is what SMEs believe the learner must learn. Should we care about the multitudinous misconceptions any particular learner might actually have? Of what value is it to try to determine (trace) which assumed productions any given individual may be using? Productions, after all, are simply constructs, and in that sense are no different from the SLT rule hierarchies proposed herein. Why do we need to worry about things like the many, many different ways in which productions may be combined to produce behavior? In the process, we also have to worry about things like changing behavior simply because we rearrange the order of productions.

Why not allow SMEs, who presumably know most about any given subject matter, determine what is to be learned—not just the steps, but also how these steps (and decisions) are to be combined? Finally, what difference would it make even if we could probe learner brains to find the exact source of the problem? One way or another we need to help the learner do the right thing, irrespective of how the behavior deviates from what is desired. To do this, the tutor needs to know what should be done, not what the learner is doing wrong.

In one sense, SLT rules are more similar to the formal constructs used in the machine learning systems built by Doug Lenat (e.g., 2006) and espoused by John Sowa (this volume). They are all constructs that can be implemented
in automated systems to produce intelligent behavior. SLT rules are unlike the other systems, however, in the sense that SLT rules can be derived systematically (and cumulatively) from given problem domains. Considerable effort has gone into the former approach, and some (e.g., Sowa, personal communication) have raised the question as to whether this approach to the problem is nearing asymptote. Since the focus of this article (and our research) is on instructional implications, it remains to be seen whether the cumulative approach proposed herein might also lead to comparable (or better) results in building intelligent systems. In either case, that question deserves serious consideration.

INSTRUCTIONAL IMPLICATIONS

As detailed above, a sharp distinction is made in SLT between to-be-learned knowledge and individual knowledge. Sets of SLT rule hierarchies represent the former. Knowledge available to individuals is represented in terms of overlays on those SLT rule hierarchies. In effect, SLT rule hierarchies make it possible to precisely represent not only what is to be learned but also what any given learner knows and what still needs to be learned at any given point in time.

(NOTE: In this sense, SLT rule hierarchies represent open learner models, which as realized in TutorIT, support student directed learning (cf. Scandura, 2005; Lee & Bull, in press).)

In effect, SLT rule hierarchies (AKA SLT content rules) provide a direct unambiguous foundation for specifying diagnostic and instructional logic. Given a set of SLT rules hierarchies, an external observer, tutor, co-learner or competitor can infer from observable behavior what an individual knows relative to given SLT rule hierarchies. Nodes in SLT rule hierarchies that an individual does not know correspond precisely to what the individual needs to learn.

Diagnosis and remediation can be specified with arbitrary degrees of precision, depending only on the adequacy of the requisite Structural (domain) Analysis (SA). An individual’s state of knowledge and the instruction needed may change dynamically. Accordingly, SLT provides a general, precise, and automatable (executable) infrastructure for developing automated learning and tutoring systems. As detailed in Scandura (2005), the structure of SLT rule hierarchies alone is sufficient to guide learning and instructional processes. Given what is to be learned, and what an individual knows (and what is still to be learned) at each point in time, provides an explicit basis for instruction (e.g., Scandura, 2005).
Representing knowledge at multiple, behaviorally equivalent levels, dramatically reduces the number of test items necessary for assessing behavior potential of any given individual. Thus, if an individual solves problems associated with any given procedural node, one can safely assume that the individual also can solve problems associated with lower subordinate level nodes. Problems associated with the latter are necessarily less demanding. Conversely, if the individual fails on problems associated with a given procedural node, then one can safely assume that the individual will also fail on problems associated with higher-level nodes.

The introduction of higher (as well as lower order) SLT rule hierarchies also provides a highly efficient basis for instruction in ill-defined domains. Higher order rules correspond to higher order relationships—with one big difference. Whereas the number of higher order relationships goes up multiplicatively with the number of nodes in the individual relationships (entering into those higher order relationships), the number of nodes in SLT representations goes up only additively with the number of rules and higher order rules (Scandura, 2005). In effect, the introduction of higher order rules dramatically reduces both the number and kinds of problems necessary for diagnosis and instruction with respect to ill-defined domains. Rules and higher order rules are strictly modular in nature. What a learner does and does not know with respect to any given SLT rule can be addressed individually.

In short, SLT provides an explicit and highly efficient basis for diagnosis and instruction in arbitrarily complex domains—e.g., where no finite set of solution rules taken individually is able to solve a sufficiently large subset of problems in the domain. SLT’s UCM together with an individual’s behavior potential (measured in terms of SLT rule hierarchies) makes it possible to predict success not only on problems solvable via individual SLT rules, but also on novel problems whose solutions require use of multiple SLT rules.

Assessment and instruction on higher order SLT rules necessarily will play a dominant role in complex domains. These higher order rules may be needed to generate new SLT (solution) rules, to gain expertise and/or to select from alternatives. In design problems, for example, multiple SLT rules may produce a satisfactory result. The key to understanding design problems depends on identifying higher order selection rules specifying conditions under which one design may be preferable to another.

The bottom line is that deciding what to teach and when to teach can be based entirely on the structure of to-be-learned SLT rule hierarchies. SLT rule hierarchies play the key role in assessing behavior potential—what a learner can and cannot do. The corresponding data structure AST on which these SLT rules
operate are equally important. The latter define problems to which these SLT rules apply. Identifying domain of applicability is especially important with higher order rules because their effects are indirect—making it easy for learners to misapply what they have learned.

**Diagnostic and Instructional Logic with respect to Lower Order Rules:**
While various proposals have been made as to how best to teach, all in one way or another attempt to move learners from where they are toward some set of instructional goals. Given the modular nature of knowledge representation in SLT, it is instructive to consider the process first with respect to single SLT rule hierarchies. To emphasize their executable nature, SLT rule hierarchies are often referred to below simply as **Abstract Syntax Trees (ASTs)**.

The approach closely follows that used by TutorIT in making diagnostic and instructional decisions (Scandura, 2005). Fundamental in TutorIT’s decision-making are the **Learner Model (LM)** and various options chosen by the author (including allowing learner control). All diagnostic and instructional decisions are based on these options together with the state of the LM.

**Learner Model (LM)**—The LM is defined as an overlay on AST-based SLT rule hierarchies (ASTs) representing the knowledge to be acquired. In the preferred embodiment, each node in an AST has one of three states: known (+), unknown (-) and to-be-determined (?). To make this concrete assume we have a well-defined problem domain in which any given problem can be solved by a single rule (e.g., column subtraction problems solved by the method of regrouping). More accurately, any given problem may be solved by interpreting/executing the corresponding AST on the given problem. In general, only some of the nodes will be traversed in the process. For example, a problem like “357-135” does not involve “borrowing” (aka “regrouping”). Corresponding nodes in traversing the AST will simply be ignored. Accordingly, if the learner gets this problem correct, the only nodes marked “+” in the AST will be those that do not involve borrowing. Additional problems will be required to promote full mastery.

Solving any given problem (by applying an AST) defines any number of sub-problems (e.g., borrowing, subtracting column, etc.). Each sub-problem corresponds to a unique node in the AST. Each such node is the root of and defines a unique sub-tree—that sub-tree whose execution solves the sub-problem. Each such node also defines a corresponding data structure (recall the close relationship between structural and procedural knowledge). Accordingly, these relationships make it possible to automatically generate sub-problems precisely when and as they are needed.
An AST containing up to 20 or so nodes appears sufficient for representing specific skills taught in most schools—at a level of detail sufficient to ensure that terminal nodes are atomic for the target learner population. “Atomic” in this context means that the corresponding SLT rule is so elementary that it can be assumed available to all learners in the target population.

**Diagnosis and Instruction**—Each node defines a particular sub-problem—because the corresponding AST can be executed interpretively. The sub-problem associated with any given node is that defined by executing the AST (e.g., using TutorIT’s interpreter, Scandura, 2005) up to the given node. Executing the sub-tree defined by the node solves that sub-problem. Each sub-tree represents one particular, executable way to solve the sub-problem—specifically, the way specified by the subject matter expert in constructing the SLT rule hierarchy.

Because SLT rule hierarchies represent the same knowledge at multiple levels of (behaviorally equivalent) expertise, the actual number of sub-problems required for diagnostic and instructional purposes may be reduced considerably. Assuming success on a sub-problem corresponding to a non-terminal node (e.g., borrowing), for example, is behaviorally equivalent to success on all simpler sub-problems. Success on a non-terminal node (i.e., marking it ‘+’) is equivalent to being able to execute all lower level nodes. Hence, all lower level nodes also may be marked “+”.

Conversely, failure on a node (marking it “-“), implies that higher-level nodes also may be marked ‘-‘. Other than by chance (mutually compensating errors), it is impossible (using the same rule) to get a harder problem correct unless one can solve all of the simpler sub-problems of which it is composed. This type of inferencing dramatically increases tutor efficiency (patent pending).

Irrespective of relationships assumed between nodes, all diagnostic and instructional decisions depend directly on the current state of the LM (i.e., the structure of AST-based SLT rules in which “+”, “-“ or “?” has been assigned to each node) and delivery options specified by the author (in AuthorIT’s Options Tool, Scandura, 2005).

These options determine which sub-problems to present, when and by whom. In general, it makes sense to choose those untested nodes that maximize the information gained about what learners do and don’t know. The purpose of diagnosis is to determine the current state of the learner’s knowledge. Hence, nodes whose status has already been determined (e.g., marked “+” or “-“) may be ignored for this purpose. Actually, this is not entirely true in TutorIT. Selected nodes may be singled out for higher levels of expertise (e.g., where the author wants the learner to know the material extremely well). “Automation” nodes are tested further even after they are marked “+” (e.g., to insure that the learner can respond correctly.
within predefined time limits). The author may also let learners ignore intermediate steps so they need not use any prescribed solution procedure.

Aside from Automation, diagnosis concentrates on nodes, marked “?”, whose correct state is yet to be determined. Nodes are chosen, corresponding sub-problems are presented to the learner, the learner responds and TutorIT evaluates that response. In addition to Tutor control, learners may be allowed to choose not only problems but also whether to be asked or shown how to solve any given problem.

TutorIT supports an extensible number of response and evaluation types for this purpose. Matching text, including wildcards, Edit and Combo boxes, and clicking in pre-set regions are directly supported, as they are in many ITS systems. In addition, a fully adequate tutoring system should allow learners to construct arbitrary ASTs, representing complex structures. While authors have this ability, learner support in TutorIT is currently limited. While learners theoretically can construct arbitrary data ASTs, for example, this is not necessarily easy to do, even more so where learners are to construct ASTs, involving operations.

Similarly, evaluation is limited to experimental work with: a) matching ideal against learner-constructed AST structures (where structural inferences may be made based on levels in data ASTs) and b) determining the best fit between responses and author-determined alternatives (used, e.g., to identify learner misconceptions—to the extent that the author believes this to be important).

When ASTs are refined to the atomic level, empirical research demonstrates that a single test item is sufficient (e.g., Scandura, 1971, 1973, 1977)—although see the next section on Uncertainty. In the case of conditions, however, all of which are necessarily terminal in AST-based rule hierarchies, one test item is required for each distinction defined by the condition. If..then and loop conditions, for example, distinguish between true and false. Case conditions can have any finite number of distinctions.

*(NOTE: Under laboratory conditions, test-retest reliabilities were 96%, with a 95% confidence interval between 93-99% (Scandura, 1977, Chapter 8). Coefficients of Generalizability (Rajaratnam, Cronbach & Gleser, 1965) based on careful a priori analysis of column subtraction ranged from .74 to .87 (Scandura, 1977, Chapter 9, with John H. Durnin).)*

Ideally, one only wants to provide instruction when and where it is needed. Providing instruction when it is not needed both wastes time and can even reduce learner interest. Hints, when given before a learner has mastered necessary prerequisites, can lead to learner frustration and giving up (e.g., Scandura et al, 1969).
Instruction ideally should be provided only when the tutor knows that the learner does NOT know the corresponding material. Accordingly, TutorIT by default provides instruction only on nodes designated by “-“. Once instruction corresponding to a “-“ node has been presented to the learner, that node is marked “?” because the tutor cannot know for certain whether or not the learner has mastered the material. This can only be assured after subsequent testing.

**Uncertainty**—The above analysis is deterministic in nature (cf. Scandura, 1971) and based on the fundamental assumption that defining ASTs are refined to the atomic level. Clearly, there are any number of reasons why not all terminals will be atomic, not the least of which is the time and other resources an author wishes to devote to Structural Analysis (SA). Learning in the context of only partial analysis may be less than optimal. Preliminary SA should be viewed as an interim step in an on-going process, a process that may be picked up and extended at any later time.

Long experience indicates that learning is better when based on even preliminary analysis, as is the case in relying on simple taxonomies (e.g., Jonassen, 2006). What might rightly be outlawed in TICL is attempting to define instruction independently of what is to be learned—unfortunately, as still is all too commonly done (cf. Scandura, 2006a). On the other hand, attempting to fit all content into a fixed set of categories can easily miss important differentiators leading to ambiguous and/or even contradictory results.

Only to the extent that this notion of incremental (and hence increasingly better defined) instruction becomes part of the working lore of instructional designers, will instruction realize its scientific potential—of increasingly precise understanding, predication and control of student learning.

Nonetheless, it is equally important to know what to do whenever SA falls short of the ideal. In this case, neither testing nor instruction can be all-or-none. Just because a learner gets one sub-problem correct, doesn’t necessarily imply success on other sub-problems associated with the same node. What to do in this case? One alternative might be to increase the number of states. There is nothing magical about just +, - and ?. TutorIT itself currently includes Automation as a distinct level of expertise. Indeed, identifying reasons why a learner makes particular mistakes has long been an integral part of ITS research (Sleeman & Brown, 1982). This option is inherent in the above discussion of “best fit” evaluation. Another approach might be to introduce probabilities (e.g., assign probabilities to nodes, or to decision making itself, Bayesian conditional decision making being one of the more obvious choices).
Although both options are quite reasonable, subsequent research has not convinced many highly experienced designers that identifying sources of mistakes, for example, is an essential factor in tutoring systems (Foshay, personal communication). Introducing probabilities also holds promise. However, decision-making and making a general-purpose tutor that works with ALL content becomes correspondingly more difficult.

Accordingly, TutorIT takes a different route, which accomplishes similar goals. One variant would require each node to be tested (on different sub-problems) some finite number of times (greater than one). Because diagnosis as detailed above is so efficient, however, another solution is to simply require the entire diagnostic-tutorial process to take place some finite number of times. Re-confirming diagnosis and remediation is similar to what engineers do (or should do) in adding a safety factor when over designing a bridge.

Because of the strong interdependencies among nodes in an AST, the number of nodes calling for instruction will decrease significantly with each iteration. Early testing at higher levels in an AST will have the same effect with advanced learners. In general, testing the equivalent of, say, 5 times on an AST as a whole will be far more efficient than it would be if each node were to be tested 5 times individually. A study comparing repeated adaptive instruction with alternative approaches, in the case of uncertainty, would be of considerable interest.

**Extension to Collaborative Learning**: Learners can and do learn from one another. And, there is significant research on the potential benefits of collaborative learning in ITS (e.g., Arroyo et al., 2004; Mirzarezaee, 2004). Although convincing evidence is lacking, some investigators might even argue that learners get more from each other than from some omnipotent tutor. Like most similar research in education, however, the ultimate answer is likely to be “sometimes”—under typically hard to define conditions.

In either case, one obvious way to extend the above model to collaborative learning is to include multiple learner models. In addition to an automated tutor, one might allow cross-over effects. One learner answering or showing how to get the right answer in a shared environment, for example, is essentially equivalent to the tutor presenting that same information. Accordingly, other learners could reasonably be assigned unknown “?” status. Just as when the tutor presents instruction one cannot know for sure whether or not the others actually learned. Diagnosis and instruction by an automated tutor could either be eliminated, or could proceed as before. In either case, learners who get an item correct (+) may
be relieved of having to answer—alternatively, their results might be put in the category of bridge-building “insurance”.

**Tutor Options:**— As detailed in Scandura (2005), authors are given considerable discretion in deciding how AST-based SLT rule hierarchies are to be delivered to learners. The author may choose to include or not to include any number of variations on Adaptive, Diagnostic, Instructional, Simulation, Practice and Learner Controlled modes. The above description obviously emphasizes Adaptive mode, with Diagnostic and Instructional modes being special cases thereof. Learner Control mode allows the learner to select problems and nodes in AST-based SLT rule hierarchies—along with whether to be tested or to receive instruction.

There are several interesting variations as to strategies that may be used by the tutor to sequence instruction. Given any problem, TutorIT begins by choosing a node that can be tested by that problem (not all AST nodes can be tested by any given problem). On what basis are these selections to be made? Perhaps the most natural order is to select nodes in the order they are executed in solving the given problem. This results in sequential instruction (and/or testing) as one might in a classroom. Other reasonable alternatives are to favor selection from the top-down, when the learners already know a fair amount about the content, or bottom-up, where the reverse is true.

Other important choices pertain to dependency on prerequisites. For example, should one teach an unknown (‘-’) node before all of its prerequisites are mastered—or otherwise? The answer depends on whether or not the author believes learners are able to take bigger leaps, and consequently to learn more efficiently. Similarly, should a learner be tested on a node irrespective of prerequisite status. A conservative strategy might require that the learner master prerequisites before being given problems that require them. At present, these options are defined by the author, or determined by the Learner, but it is not hard to envision other strategies that could accomplish similar decision making automatically (e.g., strategies based on prior learning history).

**Higher Order Knowledge:** TutorIT has been used in conjunction with higher order SLT rules—taken individually. These higher order rules were based on a taxonomy of basic mathematical processes (Scandura 1971a) motivated in part by earlier work by Dienes (e.g., Dienes & Jeeves, 1965). These higher order processes comprised three pairs of inverse processes: a) Detecting regularities from examples, and the inverse of constructing examples of regularities, b) Understanding symbolic and iconic representations of STEM content and the inverse of representing understandings symbolically and/or via icons and c) Deduction, making inferences
based on given information, and Axiomatization, identifying fundamental assumptions from which other information may logically be deduced.

This work motivated the development of a series of TutorIT tutorials. Each kind of process involves the generation of new rules, and hence by definition is of a higher order. For example, one regularity in the series 1, 2, 3, 5, 8, … may be described as “next number is the sum of the previous two”. This regularity allows one to extend the series indefinitely (i.e., 13, 21, etc.). Any number of higher order rules might be used to derive such regularities (i.e., SLT rules). One such higher order rule, for example, might simply look at successive pairs of elements in the given sequence to see if the next is the sum. Obviously, this would be a very restricted higher order rule of very limited value—it is
essentially equivalent to the regularity (i.e., lower order rule) itself. A more general higher order rule might try arbitrary arithmetic operations (e.g., +, -, x, /) with successive pairs. Such a higher order rule would work with a series like 1, 3, -2, 1 (where subtraction may be used). A still more general higher order rule would look for similar relationships between successive pairs of terms, triples, etc. Such a higher order rule would be sufficient, for example, for deriving regularities for arithmetic series, involving a common difference, and algebraic series with a common multiplier.

None of these higher order rules, however, comes close to universality; they are intrinsically context dependent. Thus, a higher order rule, which allows a knower to derive (some) regularities in number series, may be utterly useless in other contexts. Nonetheless, it is easy to envision higher order rules of this type that extend quite broadly. Certainly, experts develop such higher order skills, some to a high level indeed.

Given the documented importance of so-called “situated” learning, it should be emphasized that SA results in both SLT rule hierarchies and data structure ASTs on which they operate. Domain structures in data structure ASTs explicitly define (situated) range of applicability, and correspond to automated, perceptual (encoding) knowledge (cf. Bransford et al, 2000). Range structures in data structure ASTs define problem goals and correspond to equally automated response (decoding) capabilities.

**Extending TutorIT to Ill-Defined Domains:** TutorIT does not currently support ill-defined domains in which solving problems requires the derivation of new solution rules. Nonetheless, we have seen how Structural Analysis (SA) makes it possible to identify higher as well as lower order rules associated with any problem domain, specifically including ill-defined problem domains. We have also seen how the rules and higher order rules identified via SA operate in a completely modular fashion. The Universal Control Mechanism (UCM) is defined completely independently of any knowledge, whether it be higher order or lower order. UCM is designed to generate new knowledge or to resolve conflicts or to automate existing knowledge. In effect, exactly the same principles of diagnosis and remediation apply equally in the case of higher order as well as lower order knowledge.

Reconciling the above with other approaches to knowledge representation used in computer-based learning and instruction requires explanation. Relational networks used to represent school arithmetic (e.g., ALEKS) involve literally hundreds, even thousands, of nodes. One reason for this large
difference is obvious. ALEKS-type representations refer to large integrated relational networks of nodes, involving connections not only within specific skills (e.g., facts, subtraction and addition algorithm, etc.) but also between such skills. The above node estimates (e.g. 20) refer to individual SLT rule hierarchies as commonly taught.

A second reason is subtle, but even more important. The number of nodes in ALEKS-type networks goes up with the \textit{square} of the number of nodes associated with the various skills (to show relationships between the skills). This dramatic increase arises precisely because ALEKS requires an indefinitely large number of different kinds of relationships (Scandura, 2005, also see Paquette, this issue).

This is precisely the problem that is solved by independent sets of higher and lower order SLT rule hierarchies. As we saw earlier, arbitrary refinement is accomplished via a small finite number of refinement types. Such refinement completely eliminates the need to refine n-ary relations. Such relations serve solely as terminal conditions in SLT rule hierarchies.

Each SLT rule hierarchy associated with a problem domain, no matter how large or complex, is strictly modular. Relationships between rule “modules” are represented by equally modular higher order SLT rules. Hence, the number of nodes goes up only \textit{linearly} with the number of rules. Nothing is lost in the process.

Indeed, the introduction of higher order rules actually \textit{reduces} the number of rules required by eliminating redundant rules. Furthermore, learning higher order rules has been shown to provide a sufficient basis for solving novel problems not explicitly considered when constructing a knowledge representation (e.g., Scandura, 1971, 1973, 1977; Scandura & Scandura, 1980; cf. Polya, 1962 & Scandura et al, 1974).

\textbf{CURRENT STATUS, IMPLICATIONS AND NEEDED RESEARCH}

At this point the reader may be wondering what all this implies for current theoretical, empirical and/or real world application. Does representing knowledge as ASTs and associated SLT make a real difference? Or, are they just another way of representing knowledge and/or conceptualizing the instructional processes. After all is said and done, it often appears that there are as many theories and instructional models as there are people working in the area. Furthermore, other “deep structure” theories, such as ACT and SOAR, are much more broadly known and have been concerned with similar issues for many years.
Additionally, the call for fundamentally different (i.e., deterministic) kinds of experimental data—based on individuals rather than groups—raises the question of why (do it at all)? Although supporting research has been available and fully documented for some time, much more remains to be done. Extensive, programmatic research by subject matter experts, computer and cognitive scientists, cognitive psychologists and instructional researchers is needed at all levels—from analyzing complex problem domains to foundational experiments with individuals.

Finally and in some sense most basic: Is SLT simply a coherent curiosity with little practical value? What specifically does SLT add to understanding cognitive behavior, or in building automated tutorial, learning and/or problem solving systems?

As emphasized at the beginning, SLT’s cognitive theory is axiomatic in nature, deriving from a small set of basic assumptions. Suppose these basic assumptions are true, as suggested by available research and/or common observation (e.g., each individual having a characteristic processing speed). Does SLT offer anything substantial beyond what competing theories offer? As noted several years back by Norbert Seel (TICL in San Diego, circa 1998), for example, many instructional systems start with what appear to be different models but end up looking similar in practice.

What if anything does SLT offer beyond what is already known? And, what needs and/or should be done next? We consider each area of research in turn.

**Representing Knowledge in terms of AST-based Structures and Procedures:** Our analysis demonstrates that competence (what is to be learned) associated with any given problem domain can be represented in terms of a finite set of higher and lower order SLT rule hierarchies. Each SLT rule hierarchy represents equivalent declarative and procedural knowledge at multiple levels of abstraction. Corresponding structure ASTs represent encoded (i.e., perceived) events in the real world and/or decoded (i.e., externalized) results of internal cognition—as in typing, writing or swinging a baseball bat. AST structures (see Fig. 3) correspond to, and indeed define, observables. Conversely, SLT rule hierarchies represent cognitive processes that operate on encoded structures and generate output structures (ready for decoding).

Individual knowledge is represented relative to the SLT rules associated with any content/problem domain. As detailed above, each individual’s knowledge can be represented in either of two ways. Originally, individual SLT rules were represented in terms of paths through directed graphs in which all nodes were assumed to be atomic. More generally, individual SLT rules can be represented
at arbitrary levels of abstraction as overlays on SLT rule hierarchies. Individual SLT rules in this case are defined by mastered nodes (marked “+”) at the highest levels of abstraction in SLT rule hierarchies. Collectively, these nodes represent sub-paths through procedures at corresponding levels of abstraction.

Each individual SLT rule represents an executable procedure operating on a data structure—and corresponds to a computer program written in a high-level language that directly reflects the intended semantics. Accordingly, the behavior potential of an individual with respect to any given problem domain can be represented as a finite set of interacting “routines” (i.e., individual SLT rules), each characterized by a procedure operating on meaningful data structures.

Fully accounting for human behavior (in any ill-defined domain) requires going beyond the specifically identified SLT rules themselves. Generating new SLT rules is accomplished via higher order SLT rules whose structures include other SLT rules. These higher order SLT rules are used to generate (learn) new SLT rules as needed to solve given problems. According to SLT’s cognitive theory, all interactions among these rules are controlled by a Universal Control Mechanism (UCM).

Higher order rules in SLT correspond directly to “hard wired” control mechanisms, such as chaining or generalization [from examples], used in production and other machine learning systems. The major difference in SLT is that higher order rules are strictly modular in nature (not “hard wired”) and may be added, deleted and/or modified without affecting operation of the system as a whole.

Similarly, instead of “hard wired” conflict resolution (e.g., to choose from among applicable productions), SLT uses modular higher order selection rules (cf. Scandura, 1971, 1973, 1977). The structures of higher order selection rules in SLT contain two of more applicable rules. Resolving conflicts by selecting from alternatives plays a major role in deciding what to do in situations, such as design problems, where more than one option/rule is available.

Finally, SLT also modularizes atomatization or chunking. Automated behavior is equivalent in the sense that it produces equivalent outcomes. However, the efficiency or latency of response of the associated knowledge (SLT rule) may differ considerably. The expert also can typically perform any given task in more than one way. In this case, the input and output structures involved are paramount. The actual procedures used are of secondary importance and may be ignored.

In theories like SOAR and ACT, not to mention cognitive psychology generally, “chunking” is typically viewed as fixed and irreducible. A direct outgrowth of Miller’s (1956) classic research in experimental psychology, chunking is assumed to be a fundamental, unexplained mechanism resulting from repetition or practice.
Although easily defined, however, the underlying mechanisms by which automation works remain obscure. In contrast, automatization (chunking) in SLT is assumed to be controlled by higher order rules—this time by higher order rules that operate on SLT rules at one level of abstraction in an SLT rule hierarchy, and generate (behaviorally equivalent) relatively higher level SLT rules.

**Structural (Domain) Analysis**: As detailed above, SA is a systematic method for detailing what must be learned to master any given content domain.

Demonstrating how any structure and/or procedure can be redefined indefinitely—made increasingly precise without limit—represents an important advance in hierarchical representation. Three basic types of structural refinement (component, category and dynamic), along with their variants, and three corresponding kinds of procedural refinement (parallel; selection, loop or object, and interaction) are sufficient for this purpose.

Component (element of/is a) and category (subset/part of) have been widely and routinely used in hierarchical analysis. Invariably, however, such hierarchies must be supplemented with relationships—which do not lend themselves to systematic refinement. Dynamic refinements make it possible to dispense with relational refinements (except for irreducible conditions). Eliminating the need for irreducible relationships is important—not only because they do not readily lend themselves to further refinement but also because relations do not directly map into observables.

As detailed above, dynamic refinements play an essential role in making prerequisites operational. Their importance becomes transparent even in simple domains like computational arithmetic where reading and writing numerals are essential prerequisites.

Considerable research has been done in detailing lower order SLT rules, including highly rigorous and executable representations in arithmetic and algebra. In addition, although lacking the precision made possible by recent advances in theory and tools, a variety of domains involving higher order rules were systematically analyzed in early research. These include (lower and) higher order rules in mathematics for teachers (Scandura, 1971d), straight-edge and compass construction problems in geometry (Scandura, 1974), proof construction in high school algebra (Durnin & Scandura, Chapter 4 in Scandura, 1977), and solving Piagetian conservation tasks (Scandura & Scandura, 1980).

Although mechanisms were only partly understood at the time, experimental results associated with Piagetian analysis (Scandura & Scandura, 1980) further demonstrated the importance of automatization in enabling the meaningful definition of new classes of problems—in this case, the transition from pre- to
post-conservative behavior as defined by Piaget. Early studies (e.g., Scandura, 1971a, 1973) also have demonstrated the importance and role of higher order selection rules. Nonetheless, definitive experimental work in these areas remains for the future. The importance of such research can hardly be overestimated, however, given its potential for shedding light in areas that to date remain largely unexplored territory.

SA differs in three important ways from both traditional task and other forms of hierarchical analysis in instructional design and knowledge engineering as used in Cognitive Science. In addition to hierarchies of prerequisite tasks, a) SA emphasizes what must be learned for success (SLT Rules), b) SA supports refinement of (even complex) relationships (including prerequisites)—providing a fully integrated, executable and extensible foundation for instruction and c) SA is equally concerned with identifying higher as well as lower order knowledge, including higher order SLT rules for deriving, selecting and automating other rules.

Like SA, knowledge engineering in Cognitive Science leads directly to executable representations (e.g., production systems). Unlike traditional knowledge engineering, however, SA is highly systematic, and designed for use by subject matter experts (SMEs), not just knowledge engineers. Accordingly, SMEs must learn to use SA and/or the AutoBuilder tool (see Scandura, 2005) that helps automate the process. HLD programmers (analogous to knowledge engineers) can easily and completely independently convert the resulting SLT rules into executables.

Potential benefits of SA also exist in comparison with relational systems derived from mathematical logic (cf. Paquette, this issue). Relations between relationships invariably arise when representing knowledge in relational networks. Accordingly, the number of nodes in such networks goes up dramatically as content increases in complexity. In contrast, Structural Analysis (SA) ensures modularity combined with full hierarchical representation. Hierarchical representation dramatically reduces the amount of testing and instruction required. Complexity is further reduced because higher-level knowledge is fully modular and builds on (and assumes) equally modular lower level knowledge as data. Whereas the number of relationships in purely relational representations goes up geometrically (multiplicatively) with subject matter complexity, the number of nodes associated with modular lower and higher order SLT rules goes up only additively (Scandura, 2005).

Perhaps the most important, immediate and overlooked potential benefit of SA lies in reducing the need for costly experimentation. A long history of research in SLT demonstrates that systematic analysis (and identification) of
what needs to be learned for success in any given problem domain often has a greater effect on learning outcomes than the variables being manipulated. Although direct empirical research is reassuring, I have found repeatedly that the more precisely one identifies what is to be learned, or actually being taught, the more predetermined (and less necessary) the experimental results. Running the experiment often simply confirms what is already known as a result of systematic analysis (e.g., Scandura, 1967, 1968, 1969; Scandura & Durnin, 1968, Scandura et al, 1967).

In short, analysis goes a long way in explaining findings and could provide major payoff in a variety of settings: “Meta-knowledge is (always, often, sometimes) domain specific”—what meta-knowledge? “Scaffolding is (often) important in self-directed learning” –what is actually being taught? “Some (categories of) people (sometimes) learn (some) things better via pictures than words”—what information is actually being conveyed, and how does it relate to what is being tested? “Cognitive models (often) facilitate learning and/or transfer”—same question? Even truisms like “repetition (usually) increases mastery level“ could benefit from pinpointing the kind(s) of help given during repetition. As regards scaffolding, for example, we have found not only that higher order knowledge can systematically be identified but that it can be taught directly to good effect—enabling learners to transfer that knowledge in solving new kinds of problems (e.g., Roughead & Scandura, 1968; Scandura, 1974).

Potential applications of SA range from building automated learning and problem solving systems to highly adaptive instructional systems. Indeed, I believe that systematic SA is (or should be) an essential first step in building any such new system. AutoBuilder (in AuthorIT, Scandura, 2005) currently facilitates the process with lower order rules, but needs to be extended to accommodate higher order knowledge. Although SA may appear to be most directly applicable to school and/or college science, technology, engineering and mathematics (STEM), potential implications are constrained only by imagination of the analyst.

Explicit attention in such analyses should be given to higher as well as lower order knowledge, and their domains of applicability. Generality of the higher order rules identified will depend to a considerable extent on scope of the domain being analyzed. Some may be restricted to particular sub-classes of problems, but others may cross boundaries. There is no reason to believe, for example, that a person with strong visualization, synthesizing, analyzing, or inference making capabilities will lose those abilities in dealing with a new domain. The key is pinpointing such abilities, and their domains of applicability.
to a degree sufficient for measurement.

Whether done informally, or armed with associated tools, SA needs to be systematically applied by domain (SME) experts in increasingly numbers and kinds of domains. Explicit attention needs to be given to identifying: a) higher order SLT rules for generating new knowledge, b) higher order SLT selection rules enabling learners to decide when to use alternative SLT rules and c) higher order knowledge leading to mastery. The latter is in its infancy, but there is much more to the subject than 1960s era S-R behaviorist truisms based on repetition (cf. Fadde, 2006).

In short, SA has barely scratched the surface of what is possible—especially given recent advances in systematizing the method (of SA) and supporting technologies (Scandura, 2005). Preliminary forms of SA have been used to good effect in clarifying what is learned in a number of complex domains (e.g., Roughead & Scandura, 1968; Scandura, 1970, 1973, 1974, 1977; Scandura et al, 1974; Scandura & Scandura, 1980)—but much more needs to be done. Although applied primarily to highly structured STEM domains, for example, Structural (domain) Analysis (SA) is a close cousin to “structural analysis” as the term is currently used in linguistics—where the emphasis tends more toward syntax than semantics (e.g., see Frank 1997). SA as defined herein is equally applicable to syntax and semantics, as well as to relationships between the two. Obviously, much more can and needs to be done.

Broad-based systematic application of SA by SMEs could have major payoff, not only in reducing (not eliminating) the need for costly empirical research, but more importantly, in providing an increasingly solid foundation enabling major advances in education. A major national research effort would be extremely timely.

**SLT’s Cognitive Theory:** SLT’s cognitive theory distinguishes memory-free cognition from that including human-like factors such as response latency and processing capacity. Memory-free cognition has potentially important implications for building automated (e.g., machine) learning and instructional systems, where latency and processing capacity ordinarily are of only secondary concern. The primary difference between SLT’s memory-free theory and traditional expert systems is the complete modularization in SLT of control, conflict resolution and automation (chunking) mechanisms. Practically speaking, modularization makes it possible to incrementally enhance any given expert (or other problem solving or automated learning) system—without any change to the system’s hard-wired infrastructure.

Clearly, it is impossible at this point to know what practical advantages, if any, the proposed approach offers in comparison to the tens of millions of dollars
that have gone into well-developed alternatives (e.g., Lenat, 2006). The potential advantages, however, would seem well worth exploration. My guess is that an operational prototype system of this sort could be built within two years for about $1 million.

Where desired, one also might enrich memory free systems to include human like processing constraints. Such constraints, for example, would allow systems to forget (as the processor becomes overloaded) as well as to construct new knowledge, and to gain efficiencies associated with expertise.

Empirical support for some fundamental assumptions regarding human cognition is both direct and strong. Nonetheless, it is limited in scope and should be confirmed in other laboratories. I should emphasize, however, that such research is quite demanding. Experiments on UCM, for example, must be run under idealized conditions ensuring that the assumed higher and lower order SLT rules have both been learned and are actually available to the learner. Memory, more specifically processing capacity, must by definition NOT be a factor. “Learned” rules are retrieved from memory via other rules as in original learning—ALL rules that are not immediately available (on paper or in immediate memory) are to be derived from other rules as situations demand. The only difference between higher order derivation rules and retrieval rules is that the latter are automated versions of the former—put differently, retrieval of previously learned rules in SLT becomes increasingly automatic/expert (e.g., with practice) in comparison with original derivation.

The question we ask with regard to the UCM is quite different from that in standard psychological experimentation. UCM is assumed to be universal. Testing for UCM’s hardwired presence, therefore, requires insuring that needed rules are immediately available to the learner (e.g., supported on paper). To see why, suppose we were to program any number of lower and higher order SLT rules into a computer. What would happen? Nothing! The computer must be given some mechanism to determine what rule to use and when. Whereas UCM must clearly be programmed (in a computer), the question we ask of humans is whether UCM comes built in.

A key requirement, one that to my knowledge no other control mechanism offers, is the ability to operate interchangeably irrespective of the rules involved. In particular, UCM must accommodate without change essentially any kind of higher order rule. It must operate equally well whether a higher order rule involves the generation (or retrieval) of new rules, selection from among alternative matching rules or automation—converting relatively procedural SLT rules into equivalent ones with more complex structures.

A reasonable amount of fundamental research has been conducted on the
generation of new rules (e.g., Scandura, 1971, 1973, 1974, 1977). However, there has been relatively little on rule selection (e.g., see Scandura, 1971, 1973, 1977), and even less (empirical research) on automation (see Scandura & Scandura, 1980).

**Automation research will necessarily deal with the conversion of (largely procedural) knowledge, where data structures are simple and procedures complex, to declarative knowledge, where structures are complex and procedures simple.** Indeed, early research suggests that automation is a prerequisite for defining new problems (Scandura & Scandura, 1980). Specifically, newly derived declarative knowledge provides the foundation for defining new classes of problems. Even our analysis of simple column subtraction, for example, shows why it is essentially impossible to learn or teach column subtraction without the child first learning to read and/or write (or type) numerals. My wife and I (Scandura & Scandura, 1980) had discovered this requirement earlier in analyzing Piagetian Conservation tasks. It was impossible to train young children to be *true* conservers until they had already mastered corresponding prerequisites. For example, children did not and could not become true number conservers until they had first mastered one-to-one matching. While the importance of this observation was duly noted, providing an explicit theoretical explanation only became transparent upon realizing the central role of ASTs in representing both structural and procedural knowledge. In short, **SLT rules must be mastered (represented as high level structural mappings) before they provide a useful (even possible?) foundation for defining new classes of problems whose meaning depends thereon.** Research is needed to confirm that *chunking* is, in fact, the equivalent of converting procedural knowledge into structural knowledge—the result of higher order rules generating relatively automated SLT rules from behaviorally equivalent ones at lower levels in a SLT rule hierarchy.

There has been relatively little empirical testing of core assumptions pertaining to (individual) processing capacity, or processing speed. Relevant research will demand even more intensive training. It is essential to insure that subjects actually are using highly prescribed processes (e.g., where memory requirements are detailed).

Testing assumptions associated with an individual’s fixed capacity for processing information, for example, is very demanding. **The question here is not duplicating Miller’s (1956) classic result (7 +/- 2) with respect to groups, but rather determining whether there are fixed characteristic processing capacities associated with individuals.** Essentials require calculating precisely the memory loads associated with using given SLT Rules. These can be determined strictly
by analysis of the chunks of data (i.e., the data structures) held in memory at each point in time while solving given tasks. More demanding, one must be sure via exhaustive pre-training that the individual is actually using the assumed SLT rules in question.

Ensuring preconditions in such experiments becomes increasingly difficult with more complex SLT rules. Nonetheless, the implications of empirical results on even simple tasks (such as those by Voorhies & Scandura, Chapter 7 in Scandura, 1977) are expected to be universal and applicable to all tasks. Accordingly, processing capacity determined by experiments on simple tasks may be sufficient for determining how an individual is likely to perform on tasks of arbitrary complexity.

Identifying characteristic processing speeds will pose equally difficult experimental challenges. Speed also is directly dependent on degree of chunking/ automation. Hence, in measuring an individual’s processing speed, emphasis shifts from the number of chunks (structures being processed) to the time it takes to process any given pre-formed chunk. Common every day observation (that some people characteristically process information faster than others) appears to be the only guide for such research. In theory (yet to be confirmed), individuals may be expected to vary in the speed with which any given chunk is processed. An ideal scenario would be for ALL chunks to be processed at the same characteristic speed (for any given individual).

The important question is how theoretically to calculate speeds. One idea is to associate speed with operations on encoded chunks (together perhaps with characteristic encoding and decoding speeds). In this case, the hypothesis (of each person having a built-in processing speed) may be tested by taking the chunks as given (via training). The experimenter would then determine the time it takes any given individual to process any given chunk. To the extent theory reflects reality, knowing the chunks to be processed by any given individual would make it possible to compute response latencies for any given task. Of course, different chunks might turn out to require different processing times. In this case, the situation would become more complex but still be of interest if response times can be shown to vary depending on yet-to-be-determined characteristics of the chunks (structures) being processed. Alternatively, while processing time may vary according to chunk, processing times for given individuals may be proportionally fixed. That is, individuals may process all tasks in fixed proportion (e.g., Subject A processing at 1.2 times that of B). While such results would complicate information gathering, the results of such experiments would provide considerably deeper levels of understanding than are currently available in complex information
processing. All of this, of course, is totally virgin territory and other approaches may work as well—or better. 

So much for fundamental assumptions. There is even more to be gained by testing hypotheses derived from basic assumptions in SLT’s cognitive theory. Such research might take any number of directions. The importance of memory load in determining instructional effectiveness is one such area. Rather than trying to extract generalities from high-level abstractions (as is common in contemporary memory load research, e.g., van Merrienboer, 2006), one might do as well (or better), more efficiently by starting with an analysis of what is to be learned—specifically the memory load imposed by each condition.

More generally, SLT provides potential answers to a spectrum of heretofore unanswered (or unanswerable) questions. A study by Wulfeck and Scandura (Chapter 14 in Scandura, 1977), for example, provides highly detailed and instructive results on the role of sequencing in problem solving—the significance of which was only partly understood when the study conducted. This study was derived directly from SLT, involving both a computer simulation and associated empirical research.

This study was based on an extended analysis of straight edge and compass construction problems. SA of this domain was continued recursively until core SLT rules were reduced to three basic lower order rules (e.g., setting a compass, drawing a straight line, using a set compass to draw an arc or circle) and two higher order rules (forming the composite of two rules [= chaining] and forming a conjunction [= performing two rules in parallel]).

The lower and higher order rules identified via SA, collectively, made it possible to derive (i.e., generate) solution rules for every straight edge and construction problem. The only variable was the number of levels of derivation required to generate the needed solution rule. For example, simple problems can be solved directly using one of the initial rules. Others require application of a higher order rule to generate the needed solution rule. Still others require additional levels of generation (e.g., generating new higher order rules which, in turn, generate needed solution rules).

The simulation program (written in SNOBOL) effectively generated a hierarchy of rule sets, each set generated from the previous one by allowing exactly one level of derivation (of a higher order rule operating on lower order rules). Successively more inclusive rule sets included all rules that could be derived directly from higher and lower order rules in the previous rule set.

(NOTE: Although it was impossible in the simulation to completely separate higher order rules from control, this did not appear to impact the experimental results.)
In effect, it was possible to predetermine the number of levels of derivation necessary to solve any given problem. Moreover, it was easy to both identify and make sure that all learners entered with the assumed basic capabilities (i.e., 3 basic lower order rules plus two higher order rules; see Wulfeck & Scandura, Chapter 14 in Scandura, 1977 for details). In addition, determining the number of levels of derivation required at each point in any instructional sequence was a simple matter. In this study, problems in each sequence tested required at most two or three levels of derivation.

Deriving new solution rules imposes more demands on problem solvers than simply applying a previously learned rule. Hence, problem sequences in which relative difficulty is kept small should lead to better overall performance than those sequences in which difficulty is higher.

Four groups of 10 seventh-grade subjects each, were given different sequences of 20 geometric construction problems: Group X1 received the 20 problems arranged by the computer program so that the difficulty levels for Problems 18 and 20 were three, and for all others, two. Group X2 received a sequence obtained from the first by deleting certain problems (8, 10, 12, and 16). This increased the relative difficulty levels for Group X2 on Problems 9 and 13 to three, and decreased the difficulty level for Problem 20 to two. The higher order rule derived in solving Problem 9 facilitated solving Problem 20.

Groups R (random) and L (learner-controlled) received the original 20 problems with the first 6 problems in the same order as the other groups. After Problem 6, the problem order in Group R was random. In Group L, the individual subjects were allowed to choose which (of problems 7 through 20) to attempt next.

Subjects were run individually (under non-idealized conditions). The problems were presented to the subjects on separate sheets of paper one at a time. Subjects were given a pencil, compass, straight-edge (not a ruler)—and were required to show their work on the problem sheet.

To help insure correct interpretation of the problems, each problem statement (when presented) was read aloud by the experimenter, given elements were pointed out, relational terms (e.g., parallel) were explained, and a sketch of the goal figure (in required relationships to given elements) was drawn. If a problem was failed, the subject was shown a solution rule for it, and was required to execute the rule correctly on the problem page. To help ensure continued availability of derived solution rules, subjects retained the problem pages and were allowed to refer to them as desired. In effect, differences among the treatment groups may be attributed to differences in the higher order rules learned—not to the memory of previously derived lower order rules.
All subjects were initially presented with Problem 6 (to provide a base line) and all failed. Next, all subjects solved Problems 1 through 5 in order (and then succeeded on 6) indicating the corresponding 3 basic and two higher order rules were at an appropriate level of difficulty.

Mean percent success on problems (after Problem 6) for Groups X₁, X₂, L, and R were 85 percent, 73 percent, 47 percent, and 38 percent, respectively. All except X₁-X₂, and L-R were highly significant. Evidently, under these experimental conditions, problem sequences in which problem difficulty is kept relatively low, as was the case with sequences X₁ and X₂, lead to significantly better performance than do random or learner-controlled sequences. Furthermore, subjects must have used previously derived rules in generating solutions to later problems. If sequence played no role, then all groups would have performed similarly.

More interestingly mean times to solution on specific problems common to Groups X₁ and X₂ were in complete agreement with predictions from SLT. The only significant differences in solution times across Groups X₁ and X₂ occurred in predicted directions on Problems 9 (X₁ < X₂), 13 (X₁ < X₂) and 20 X₁ > X₂). That is, X₂ Ss took longer to solve problems where three levels of derivation were required (on problems 9 and 13). Conversely, X₁ Ss took longer on Problem 20 because they did not previously learn the higher order rule, which made the problem easier for the X₂ Ss. These differences are what one would expect based on the number of levels of goal switching (i.e., levels of rule derivation) required during problem solving. X₁ subjects evidently were able to retain and use higher-order rules derived on some problems (9 and 13) on later problems (20) (despite not being given memory support for higher-order rules). In short, X₁ and X₂ subjects learned different higher order rules on earlier problems enabling each of them to more quickly solve a particular problem that the other group could not. This differential performance was directly attributable to the different number of levels of derivation required to solve the later problems. In each case, the group requiring three levels of derivation performed more poorly than the group requiring only two.

The role of control (i.e., UCM) was further highlighted by the fact that no subject solved any problem where the required difficulty level was greater than three (72 cases). Presumably, memory load approximated subjects’ processing capacities at step sizes around three.

To extent that rule recency, or Einstellung, is involved in Ss’ processing, an even closer coordination between predictions and theory might be expected where limitations on processing capacity are built into a simulation. In this case explicit attention would be given to exactly what rules each subject had available at each stage of problem solving.
Finally, individual subjects in Group L performed very differently. Those who chose next problems based on problem difficulty had correspondingly greater success. Three subjects stated that they chose on the basis of similarity (of problem statement and display) to previous problems. The problems they chose tended to be of low difficulty (most often at level two). About 73 percent of these problems were solved. In contrast, two subjects chose dissimilar problems, with relative difficulty levels exceeding two. None of these problems were solved. (The remaining subjects indicated no particular basis for selection, and solved about half of the problems they chose.) These results and others (e.g., Pask & Scott, 1971) suggest that some subjects may have useful problem selection skills (rules), whereas others do not. Such results further highlight the need for research on rule selection.

More generally, this study demonstrates a rare level of theoretical prediction and control on specific complex tasks. Accordingly, one can envision automated systems based on similar assumptions with unprecedented levels of instructional effectiveness (cf., Scandura, 2005).

**Assessing Knowledge Potential:** As is well known, individual differences in experimental research are typically much greater than group effects. As noted by the late Bill Winn (reviewed in Scandura, 2006a), for example, “while even dyslectic and normal children performed similarly as a whole, individual differences were much greater.” The key to understanding such results often lies in precise identification of what needs to be learned and what any given individual already knows relative to that knowledge. It would not be surprising, indeed might be expected, that prior analysis would suggest cognitive demands unrelated to known dyslectic deficits.

As summarized in Scandura (2006a):

“A key question ... is the extent to which pre-analysis of the content domain may help to understand individual differences, eliminate ambiguity and increase the precision with which one can predict behavior. Such analysis may be done informally via operational descriptions and/or examples, as we did back in the 1960s, more precisely via directed graphs (flow charts), as in the 1970s, or finally and most rigorously in terms of AST-based (SLT) rules. From an instructional perspective, perhaps the single most important result of SLT research over a 40 plus year period is that the more precisely one can identify what must (or might) be learned in any given situation, the better job one can do of assessing what the learner does and does not know—and of teaching it.”
At this point, the method used in SLT to assess behavior potential is well defined. Given a set of to be learned SLT rules, it is possible to efficiently identify what any given individual does and does not know relative to each such rule. The arbitrarily detailed nature of SLT rule hierarchies (representing multiple levels of procedural and structural abstraction) provides an explicit basis for drawing inferences about untested nodes (from tested ones). Hierarchical relationships between nodes at different levels of abstraction dramatically reduces the amount of testing (and instruction) required.

The higher the level of the node in question, the less important are the actual (procedural) steps the learner uses. At higher levels of abstraction the knowledge is more structural in nature. Assessment at higher levels is based increasingly on the learner’s ability to perform (i.e., generate correct answers), and decreasingly on the method used to construct the answers. AuthorIT (Scandura, 2005, www.scandura.com) differentiates levels of expertise by making it possible, both to add time constraints on responses and/or to omit intermediate steps. Skipping procedure-specific steps when evaluating responses allows learners to solve problems differently than prescribed in the associated SLT rule hierarchy. Distinguishing levels of expertise is certainly one area where more research is needed.

At the other extreme, deterministic precision requires that terminal operations (and corresponding low level data structures) make contact with prerequisites available to all intended learners. Data obtained in conjunction with well-defined, highly structured content (e.g., Durnin & Scandura, 1973) were highly reliable and consistent. Success or failure on single test items associated with any given path (equivalent to a node in an AST) was almost always sufficient in determine future performance.

Limitations of time or resources, of course, particularly in analyzing more complex or less structured domains, may not always allow this degree of analysis. An alternative approach may be used in this case. Multiple successes may be required to demonstrate mastery on any given node when behavioral atomicity is not immediately achievable. Doing so amounts to adding a safety factor, much like an engineer might do in designing a bridge.

Confirming success on each node multiple times is not nearly as demanding (in test time) as it might appear initially—precisely because AST structures support strong inferences from success or failure on any given node. These inferences make it possible to infer mastery concerning entire subtrees of nodes (either above or below the one tested). This type of inference is straightforward and simpler than introducing probabilities and correspondingly more complex decision-making. The only requirement is that the AST in question accurately reflect content dependencies.
It would be desirable to compare this repeated hierarchical approach to testing with more traditional stochastic (e.g. Bayesian) approaches based on probabilities. Determining the advantages and limitations of multiple testing of individual nodes in ASTs would have broad implications and should be done.

Another area where research is needed is in assessing higher order knowledge. What makes such assessment challenging is that input and/or output data necessarily include SLT rules, not just static data. Currently available technologies (in particular SoftBuilder, www.scandura.com) make it possible to construct the necessary higher order rules. The processes for doing so, however, are far from automated, making the task demanding, even more so for computer-challenged learners. Extending AutoBuilder to support higher order knowledge construction in SA will necessarily be an important first step in this direction.

Even where higher order rules can be defined, AuthorIT’s Blackboard Editor also will have to be extended to support problems having SLT rules as inputs and outputs. One can easily support the dynamic nature of such inputs to higher order rules using available media rich technologies such as Flash. Allowing learners to construct SLT rules as outputs during testing will be more difficult. Prototype construction technology currently exists. In particular, AuthorIT (and to a lesser extent TutorIT) currently supports allowing users to construct responses (questions). Nonetheless, making these capabilities accessible, particularly to young learners, poses considerable technical challenges. While neither extending AutoBuilder or the Blackboard Editor is exactly trivial, however, both do appear feasible given necessary resources.

**Automated Instruction:** Clearly, the close relationship between individual nodes in SLT rule hierarchies provides a rigorous basis not only for diagnosis but also for making detailed instructional decisions. In addition to strong inferences concerning untested nodes, the resulting learner model specifically identifies what needs to be learned and/or taught.

The ease with which delivery methods can be configured in AuthorIT and delivered by TutorIT makes it practical for the first time to systematically compare multiple variations on alternative pedagogies—all in the context of fully automated systems. Available technologies make it possible to detail theoretical hypotheses with a degree of precision previously unattainable, thereby exposing to empirical test any number of questions that have defied clear answers. Instead of inconclusive debates, we can envision hard data supporting constructivist, instructional design, adaptive and/or other pedagogies. Such research would deepen insight into the many claims and counter claims so widely disseminated in the field of IT—and is long overdue.
Pedagogical logic associated with individual SLT rule hierarchies has already been implemented in TutorIT. What remains to be done is to extend TutorIT to accommodate ill-defined domains, where problem solving involves interactions among higher as well as lower order SLT rules. This is an exciting area overflowing with both research and practical opportunities. The Wulfeck and Scandura study (1977) cited above barely scratches the surface of what is possible.

An essential prerequisite will be extending available technologies to deal with ill-defined problem solving. To date, the only options have been to rely on production system based ITS and learner directed systems. The debatable fit between biologically inspired theory and instructional needs has led to complications making development expensive. A major alternative, emphasizing self-directed learning, has other limitations. While launched with much hope, it didn’t take long to rediscover the importance of proper guidance (now called “scaffolding”) during the learning process (cf. Scandura, 1964). The present approach to KR and associated theory offers a third approach with seemingly unlimited potential.

AuthorIT must be extended to support the representation of higher order knowledge. In addition, it will be necessary to extend TutorIT, enabling it to solve unanticipated ill-defined problems (by deriving and/or selecting from among alternative solution SLT rules). Automating instruction in domains, where problem solving cannot be reduced to a fixed set of solution rules, requires the ability to derive and/or select from among possible rules as needed. In short, TutorIT must at minimum be supplemented with UCM. Without this essential capability, TutorIT will be unable to solve and hence evaluate learner responses to unanticipated or otherwise complex problems (e.g., constructed by a learner)—much less to provide the guidance (scaffolding) needed in open-ended systems.

(Note: In fact, TutorIT does support simple chaining but this is hardly novel and quite distant from the planned potential.)

A general solution to this problem would require building an automated learning and problem solving system that scales to match or exceed the variety and complexity of existing systems, such as SOAR (e.g., Newell, 1990) and Cyc (Lenat, 2006). Given the resources devoted to those systems this would not be an easy challenge. The needs with respect to instructional systems may not be as demanding for two reasons: a) content typically taught is normally quite constrained and b) neither SOAR nor Cyc purport to address educational issues.

Production systems, ACT theory and relational networks, on the other hand, have been intensively used in building advanced ITS systems for many years. Systematic comparison with have these approaches is long overdue, and an
important goal of this special issue on Knowledge Representation, Associated Theories and Implications of Instructional Systems to be followed by open to-be-published commentary and dialog.

REFERENCES


McIntyre, Michael E. Lucidity, science, and the arts: what we can learn from the way perception works. Keynote lecture to the fourth symposium on human development, Kobe, Japan (published in bulletin of the faculty of human development, Kobe University, 2000, volume 7, number 3, pp. 1-52).


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


Scandura, J.M. Problem solving in schools and beyond: Transitions from the naive to the neophyte to the master. *Educational Psychologist*, 1981, 16, 139-150.


