Instructional Science **29:** 311–336, 2001.

Structural Learning Theory: Current status and new perspectives[1,2]

JOSEPH M. SCANDURA
MERGE Research Institute, Scandura.com and University of Pennsylvania, 1249 Greentree, Suite 100, Narberth, PA 19072, USA (E-mail: scandura@scandura.com)

Abstract. This article presents the current status and new perspectives on the Structural Learning Theory (SLT).[3] Special consideration is given to how SLT has been influenced by recent research in software engineering, and the range of possibilities it opens for instructional research and practice in the twenty-first century. Starting with fundamental precepts of the instructional process, a generalization of the SLT is proposed that offers an integrated, parsimonious, operational and predictive (as well as explanatory) account of competence, cognition and behavior potentially from birth onward, and their implications for instruction. Supporting examples and experimental research are cited in context.

Keywords: cognition, constructivism, instructional theory, instructional technology, instructional design, task analysis, structural learning theory

Overview

Instructional systems are conceptualized in terms of interactions between a tutor and a learner (more generally between any two or more individuals) with respect some shared content domain (Figure 1). The basic question is how knowledge changes over time as a result of these interactions.

SLT attempts to answer the following basic questions and derives from the associated basic assumptions:

1. What does it mean to know something, and how can that competence be represented in a way that has behavioral relevance? Exactly what is observable behavior in the first place? No matter how complex the domain, the competence required to successfully perform tasks in that domain is represented in SLT in terms of finite sets of higher and lower order rules. These rules (defined in the next section) are called SLT rules to distinguish them from the more familiar production rules. The tasks themselves are represented in terms of input-output structures.
2. How do learners use and acquire knowledge? Why can some people solve problems for which they seemingly have all of the requisite knowledge, whereas others can not? Why do people often act in predictable ways

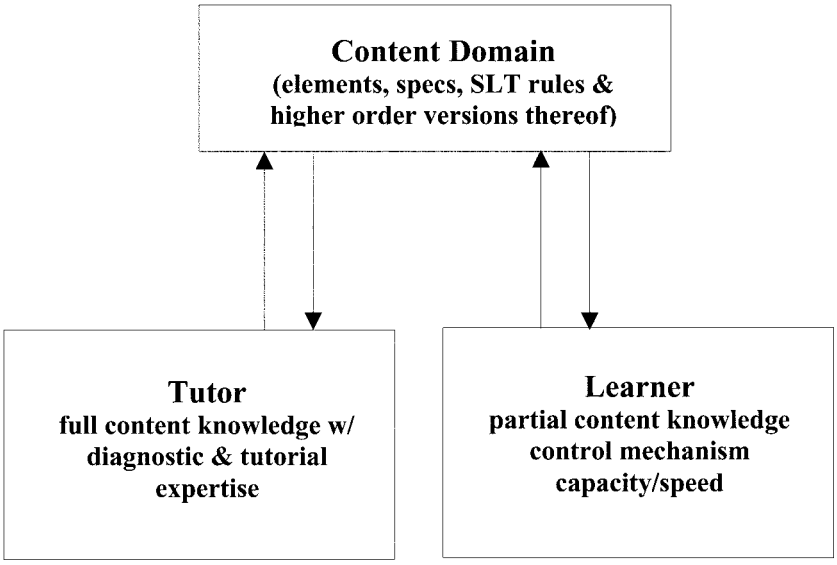

Figure 1. Key elements and constructs in the instructional process.

even when they have many different options available? And, how is it that some people solve a problem almost immediately whereas others have to labor at the problem? Most would agree today that people have minds and in that sense construct their own knowledge. In SLT, people are assumed to solve problems and acquire knowledge by constructing (learning) new (SLT) rules, the latter (in oversimplified form) being accomplished by applying higher order rules to lower order rules.

3. How does one determine what an individual does and does not know? Behavior is the only thing that instructional scientists can observe. It is impossible to know all or exactly what any person does or does not know that causes him to behave as he does. Instructional scientists do not have license or the means to look inside. It is extremely unlikely, for example, that observing brain scans, even if legal, would ever provide the kinds of information necessary to predict how an individual might perform on any given task. Explaining complex human behavior requires higher level theory. In SLT, rules of competence are used as a standard for measuring individual behavior. Individual knowledge is operationally defined in terms of behavior judged relative to those competence rules.
4. How does knowledge change as learners interact with an external environment? How do initially naïve learners gradually come to acquire mastery? What are the fundamental processes involved in longer term developmental changes? SLT attempts to answer these questions.

The remainder of this paper elaborates on the ideas and introduces new software technologies supporting the construction of instructional systems based on SLT.

Structural learning theory

Paralleling the initial four questions, SLT includes: (a) a formal way to represent content as well as a formal method for constructing such representations, (b) a cognitive theory, (c) methods for operationally defining human knowledge in terms of behavior and (d) a theory of instruction by which external agents (e.g., tutors) may influence learning.

Theoretical constructs

The following theoretical constructs are prerequisite to understanding SLT.

Observable constructs. Specifying observable behavior might seem so obvious that it hardly needs elaboration. This may be true with computers where all inputs are reduced to ASCII characters, keystrokes and (e.g., mouse) events. It is *not* true in talking about human behavior. What serve as effective inputs and outputs depends inextricably on what can safely be assumed to *act* as inputs and/or outputs for the population of learners. (In the following exposition the term *Ss* is used for learners when referring to experiments.) An observable input or output (I–O) element in SLT can be anything, concept, process or even idea. Room, number, a spoken phrase, the SLT itself are all possible I–O. The primary requisite is that input or output must be atomic (directly perceived) elements.

Hierarchical Representation. Any I–O element can be represented as a hierarchy of elements at multiple levels of abstraction. That is, the same element can be represented as an atomic whole, or in terms of the components, categories or operations of which it is composed. Table 1 shows an I–O hierarchy of variables and their values at various levels of refinement. *Room*, for example, is more fully defined in terms of its components, *bed* and *carpeting*. *Carpeting*, in turn, is further decomposed into one or more scatter *rugs*.

In SLT I–O behavior can be described at any number of levels of abstraction. An essential constraint is that behavioral equivalence must be preserved at all levels of abstraction. In this example, *room* being *presentable* is precisely equivalent to *bed* being *made* and *carpeting* being *clean.*

Problem domain. Problem domains in SLT are defined as sets of I–O pairs, including outputs without specified inputs, that happen to be of interest to

Table 1. Example of an I–O hierarchy of variables and their values at various levels of refinement

INPUT	
room <presentable, unpresentable>	*– component refinement*
bed <made, unmade>	*– atomic*
carpeting <clean, unclean>	*– prototype refinement*
rug <clean, dirty, messy & dirty>	*– atomic*
OUTPUT	
room <presentable>	*– component refinement*
bed <made>	*– atomic*
carpeting <clean>	*– prototype refinement*
rug <clean>	*– atomic*

an outside observer (e.g., tutor). Problem domains may be well-defined or otherwise (i.e., ill-defined).

Well-defined I–O domains correspond to behavioral objectives (specifications in software engineering) and are semantically meaningful sets of inputs and outputs in which there is a unique output for each input. Simple examples include:

column subtraction problems → differences
verbs → participles (ing endings)
[real world object(s) + real world action] → 'subject' 'verb'

Operations. Operations in SLT are processes, called SLT rules, which when carried out on given inputs in a domain generate unique outputs:

column subtraction
add 'ing' and drop the final 'e' (as appropriate)
write the subject (name of object), then the verb with a 'matching' ending.

Like I–O elements, operations (also called SLT rules) may be represented as hierarchies at multiple levels of abstraction. The above I–O hierarchy represents *room* at various levels of detail, both before and after some action (i.e., operation) has been performed thereon.

Such operations in SLT are called SLT rules – to distinguish them from simpler condition-action (production) rules. SLT rules are represented as triples: Domain, Range (collectively an input-output Plan) and an Operation (or Procedure).

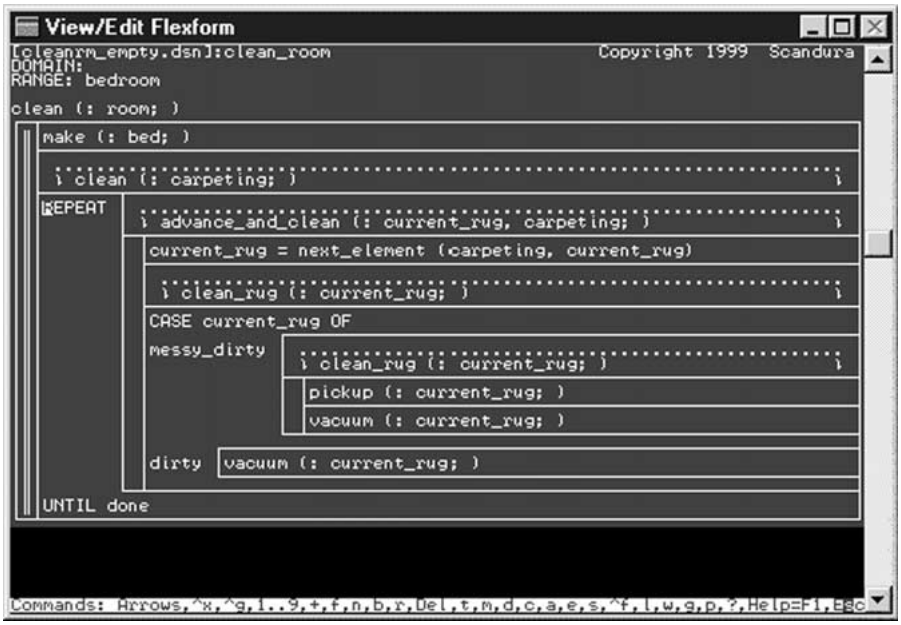

Figure 2. A sample hierarchy of SLT rules for cleaning rooms.

Figure 2 shows a hierarchy of behaviorally equivalent operations for cleaning rooms. There is an inverse relationship between the level of operations in this hierarchy and the complexity of the data structures associated with the corresponding parameters (in each operation's domain and range). For example, the *clean* operation operates on *room* as a whole, where *room* represents the full structure shown above. The lower level rule *make* operates on *bed* which in this example is a simple (i.e., atomic) element.

Problem. A problem consists of a given input structure, in which variables in the structure have been assigned values (e.g., *rug* is *dirty*), and the goal consisting of a data structure to which values are to be assigned (typically derived by applying operations). A Plan is a specification (as in software engineering) and is similar to a problem except that no input values are assigned.

Higher order rules play a central role in SLT. A higher order rule is a rule in which (some) inputs and/or outputs are themselves rules. One higher order rule, for example, might construct rules for solving different kinds of geometry construction problems from more basic operations (e.g., using a straight edge and compass, see Polya, 1962; Scandura et al., 1974b). Another (in the mind of a great 19th century inventor) might convert (some domain

of) manual operations (e.g., using brooms for cleaning) into an analogous one using electric power (e.g., vacuum cleaner).

Higher order problems and plans (specifications) also play a role in SLT (U.S. Patent, Scandura, 2001, in press).

Content domains

The content associated with any given domain is defined in terms of observable behavior and the competence necessary to account for such behavior. As illustrated below, behavior is represented as hierarchies of inputs and outputs. Similarly, competence is represented as hierarchies of behaviorally relevant rules.

Content is not restricted to well-defined domains. Ill-defined domains are arbitrary sets of I–O pairs, where the same input (in different contexts) may be paired with different outputs, where no inputs may be specified and/or where no one solution rule is known to exist. Many real world domains are of this sort: (a) chess positions and corresponding moves, (b) poems covering given topics and, (c) mathematical theorems and proofs.

To understand human behavior in such situations, it is essential to know what kinds of competence may be required. To better appreciate the approach taken in SLT, consider Artificial Intelligence (AI). In AI, the basic approach is to systematically search through a space of possible next (or previous) operations, trying and/or rejecting them, in turn, until a solution is found or the search space has been exhausted. The problem has been likened to constructing a bridge across a river on a foggy day, not knowing whether the river has made a turn. This strategy cannot succeed in its simplest form because the number of possibilities grows too quickly with the depth of the search space. While various heuristics may improve search efficiency, the methods used to identify such heuristics have been largely ad hoc.

SLT attacks the problem differently. Rather than asking how to combine a given set of components to find a solution, SLT asks, "What kinds of components are needed to solve problems (in the domain)?" Content analysis in SLT is based on a systematic process, called *Structural (cognitive Task) Analysis (SA).* Unlike 'knowledge engineering' in AI, which focuses on domain specific knowledge, SA is equally concerned with (e.g., higher order) rules that are domain independent. SA is necessarily carried out with some population of Ss and/or context in mind.

1. The first step in SA is to define the domain, in particular the observable inputs and outputs characterizing that domain. These inputs and outputs are represented as hierarchies. Identifying I–O is not always a trivial task. In teaching a non-swimmer to swim, for example, it is not immediately obvious that the inputs and outputs refer to operations. In this case, the

task may be defined in terms of converting one (already known) operation – namely floundering in the water – to another (i.e., moving through the water while staying afloat by kicking and stroking with one's arms). Viewed in this way it is clear why starting the beginner with a 'float' is often more efficient than starting without. There is less to learn! The learner never has to learn the equivalent of what is called the 'dead-man's float' because the float serves to keep the person above water. Learning to kick and stroke serves the dual purpose of keeping the person afloat when the aid is later removed.

It is instructive to consider the role of (e.g., spatial) images in this context. The basic question is how to represent continuous reality in digital form. The first thing to observe is that images can also be represented hierarchically. Ignoring retinal imprints (and their associated physiology) images may be viewed as icons – symbols which share features in common with the reality they represent (e.g., Scandura, 1970). Such icons correspond to the top level in a hierarchy, representing say, a soccer field and the players on it. Lower levels in the hierarchy represent more detail – the players, position of the ball, etc.

Domains may even be specified entirely in terms of outputs. Consider situations where the inputs in question can only be determined in terms of the solution procedures used. Domains characterized by creative tasks, such as proving (or disproving) mathematical theorems, provide a case in point. Such proofs typically are based on large numbers of (input) assumptions and other theorems (lemmas, etc.), and often involve inordinately large search spaces. Other examples include chess where the same goal may be achieved from a variety of chess positions. Although outputs may be highly abstract in nature (e.g., writing an 'interesting' poem), specifying outputs in a domain is a necessary minimum in SLT for predictive purposes. Otherwise, one is reduced to observing and reporting behavior after the fact.

2. The second step in SA depends on whether or not the problem domain is well-defined. If so, one or more solution rules (represented as hierarchies) may be constructed. If the domain is ill-defined, then one must first extract well-defined sub-domains of problems in the domain to serve as the starting point. Each well-defined sub-domain, in turn, will have at least one solution rule.
3. A hierarchy of rules is constructed for each well-defined domain. Highly systematic methods have been developed for this purpose.[4] Suffice it to say that each 'slice' through a given competence hierarchy represents one of a set of behaviorally equivalent rules. Each rule in the hierarchy generates behavior equivalent to that associated with every other one.

The only difference is in the abstraction level of the operations involved and the complexity of the data structures associated with input and output parameters of the operations. Higher level rules operate on correspondingly more complex parameters. This inverse relationship is illustrated by the I–O data structures above (see *Hierarchical Representation* in Table 1) and the clean room rule hierarchy shown in Figure 2. At the top of the hierarchy, for example, the *clean* operation in *clean (room)* is simple (and highly abstract) while the parameter *room* has a relatively complex structure. The next level operations, *make (bed)* and *clean (carpeting)* provide more detail as to process but correspondingly operate on less complex structures.

Rules for solving prototype problems in the well-defined sub-domains represent a first level characterization of the competence associated with the given problem domain. Starting with only the prototypes, however, typically leaves many 'gaps' in complex domains. Many problems (I–O pairs) are generally not covered by any prototype.

4. SA offers an explicit way to fill those gaps. This is accomplished by first converting each rule into a higher order problem whose goal includes that rule. Higher order problems, in turn, are generalized to I–O specifications, or Plans, by ignoring the originally given values. Specifically, the higher order plans include previously identified rules in their ranges and/or domains.
5. A higher order rule is then constructed for solving problems specified by each higher order plan. In general, a higher order rule will not only generate the previously identified (lower order) rule, but other rules as well (depending on values assigned to input data elements). In effect, introducing higher order rules serves to fill gaps (in ill-defined domains).

Furthermore, rules that can be generated from existing rules become redundant and may be eliminated from the set of rules characterizing competence (underlying the given domain).

This process of converting rules into higher order problems and constructing higher order solution rules may be continued indefinitely. At each stage, higher order rules become inputs and/or outputs for still higher order plans and rules. Empirical research (e.g., Scandura, 1984a) demonstrates that the introduction of higher order rules, and the elimination of redundant ones, has two beneficial effects: (a) Individual rules (including higher order ones) tend to become simpler and (b) the collective generating power of the rules becomes greater. That is, greater varieties of problems in the original domains can be accounted for via the identified rules. Repeating the process makes it possible to account for arbitrarily and sufficiently broad ranges of tasks in the problem domain with sufficiently small sets of higher and lower rules.

Discussion. There are indefinitely large numbers of real world domains. Analyzing such domains would correspondingly require an indefinitely large amount of work – especially if one had to start SA from the beginning with the introduction of each new domain. In fact, this is *not* necessary. SA is a strictly cumulative process. Many rules, particularly higher order rules, are applicable across multiple domains. Rather than having to start over with each new kind of problem, one can build on the results of previous SA: Simply add the new problems to the old domain, introducing new prototypes as necessary. The process of SA continues as before.

To summarize, competence underlying given I–O domains is represented by a set of higher and lower order rules, where each rule is represented as a hierarchy. Each hierarchy represents an equivalence class of rules, each representing a different level of automation. Given any domain characterizing the behavior of interest, SA offers a systematic means of constructing competence for solving 'sufficiently broad' sets of problems in the domain.

The same domain may be analyzed from any number of perspectives. That is, any problem in a given domain may be solvable in any number of different ways (using rules associated with the differing perspectives). The fact that European students are commonly taught to solve column subtraction problems by equal additions while American students use borrowing is well documented in the literature (e.g., Durnin & Scandura, 1973). In practice, the number of different perspectives required for instructional purposes is usually quite small. In most domains, small numbers of alternative rule sets are sufficient to account for the behavior of most learners.

In performing SA, it also is important to recognize that what serve as inputs and outputs may vary across apparently similar sub-populations of learners. Different sub-populations, for example, may make differential use of spatial versus non-spatial inputs in direction finding.[5] Similarly in soccer, for example, a valid representation will depend on whether the field is viewed from the perspective of a coach (with all players viewed from a common perspective) or one of the players. In SLT, specifying what are the effective inputs and outputs often goes a long way in defining what is to be learned.

SA has been used in practice for many years, and numerous examples exist in the literature (Scandura, 1977c, 1982, 1997; Scandura & Scandura, 1980, 1999). These include both broad-based comprehensive analyses (Scandura et al., 1971; Scandura & Scandura, 1980) and smaller more intensive analyses (e.g., Scandura et al., 1974b; Wulfeck & Scandura, 1977). SA has evolved considerably since its inceptions in the 1960s through the early 1980s (e.g., Scandura, 1964a,b, 1977c, 1984b; Scandura et al., 1974b; Scandura & Scandura, 1980). Indeed, the method itself had become increasingly systematic and repeatable over the years, leading today to precise automated

methods in software engineering (U.S. Patent Scandura, 2001, in press). Other early references of interest include (Roughead & Scandura, 1968; Scandura, 1968a,b, 1969a,b,c, 1971b, 1972).

Cognition

The author's epistemological assumptions are constructivist in nature. Knowledge acquisition is an active process. People are assumed to have minds; they learn and otherwise construct new knowledge. Unlike most constructivist theories, however, SLT is deterministic in character and firmly rooted in behavior (Scandura, 1971b, 1973, 1977a,b,c, 1978a,b). In the SLT, cognition has always been characterized in terms of lower and higher order rules operating under universal constraints. Lower and higher order rules (roughly speaking) correspond to domain-specific and general-purpose knowledge, respectively. On the other hand, all rules have the same form. The only difference between lower and higher order rules is that the latter contain other rules in their domain or range. Specific rules of knowledge correspond to one [or more] specific levels in a rule hierarchy.

The use of and interactions between known rules are governed by a universal control mechanism, and constrained by an assumed fixed processing capacity and a characteristic processing speed. In its original form (e.g., Scandura, 1971b, 1973) control was characterized in terms of goal switching: When faced with a problem, the learner is assumed to search the set of rules in his processor (often called short term memory) to see if one applies. If so, the rule is applied.[6]

If not, control is assumed to move to the higher level goal of deriving a rule that does apply. The higher order rule is then applied, generating a new rule, with control reverting to the next lower level. To wit, faced with a problem for which no solution process is known, attention focuses on constructing one. The process is assumed to continue until memory limits are readed or an appropriate rule is found.

This control mechanism was shown to work both when no rule applies as well as when more than one rule applies (Chapter 8 in Scandura, 1973). Later, it also was shown to account for automatization processes whereby more efficient rules are derived from known rules (Scandura & Scandura, 1980). Supporting experiments were conducted under a wide range of conditions (e.g., Scandura, 1971a, 1973, 1974, 1977c; Scandura & Scandura, 1980). Many of the experiments were run under carefully controlled conditions, where it was clear exactly what Ss knew on entry into the experiments and where memory was explicitly eliminated as a factor.

Although suggestive, goal switching is extremely difficult to formalize in a way that is completely independent of the higher order rules involved.

Wulfeck and Scandura (Scandura, 1977, Chapter 14), for example, were able to successfully simulate the growth of problem solving behavior over time. Goal switching control and core higher order rules, however, were subtly but inextricably intertwined in the simulation (program).

The search for a way to formally represent goal switching, with control completely separate from higher order rules, has been long and hard. Rather than goal switching as such, it gradually became clear that each level of 'goal switching' might better be characterized in terms of searching the (input) domains and ranges of (higher order) rules for (lower order) rules that match given problems at different levels of embedding.

This mechanism first looks for available rules that directly match the problem goal. When no such rule (or more than one) is found, the search moves to *rules whose outputs include rules that directly match*. The process of seeking deeper levels of matching continues until some predetermined level is reached. (In humans, this level corresponds to processing capacity associated with short-term memory.) Whenever a unique match is found, the rule directs a search for inputs that satisty its domain. If the domain is satisfied, the rule is applied. Otherwise, various options are available (e.g., needed rules [data] may be added to the base set, or a new sub-problem may be defined with the process continuing as above). Thus, if no solution rule is found for solving (i.e., containing) the given problem, then the search continues for rules whose outputs contain such rules. In the case of failure, the search moves to deeper levels. Each search level seeks rules whose ranges include needed solution rules (at some level). The recent patent (Scandura, 2001, in press) also shows how searching for plans (rules without operations) and higher order plans plays an important role in accounting for complex problem solving – e.g., the commonly observed human behavior of formulating (new) plans before their implementation.

Other studies offer support for the assumption that each individual has a fixed processing capacity (e.g., number of rules that can be kept in mind at one time) – irrespective of the task in question (Voorhies & Scandura, Chapter 7 in Scandura, 1977). This result represents an extension of Miller's (1956) classic result indicating that people on average can process between seven plus or minus two chunks of information. Support for a fixed processing speed is more hypothetical and an area in which definitive research is needed. Nonetheless, everyday observation suggests that individuals differ in the rate at which they process information. Some people characteristically talk faster than others, for example, while others are more characteristically deliberate. Indeed, such information is commonly used, often informally, in assigning people for various tasks. Research is needed to determine how much of such behavior is due to innate characteristics and how much to what has been

learned. One hypothesis, for example, is that (given otherwise equivalent intellectual powers) faster innate processing leads to greater numbers of more specific rules whereas characteristically more deliberate behavior leads to more highly structured general-purpose higher order rules.

To summarize, the kind of behavior associated with any given problem necessarily depends on what the learner knows at the time. Problem solving is necessary when no solution rule is immediately available. Higher order rules are assumed to play a key role in deriving solution rules. Behavior is simpler when the learner already knows a solution rule. Learning does not stop there, however. Other higher order rules are used for automatization. Procedural complexity in rules is successively replaced with structural complexity as behaviorally equivalent rules in a competence hierarchy are constructed (learned). The more complex the domain and range structures of a solution rule's parameters, the more rapid and more flexible the behavior. Automated rules correspond to rules at (or near) the top of the competence hierarchies to which they belong (see *Content*).

Top level rules (in competence hierarchies) are atomic and, effectively, are operational versions of what is commonly called *declarative knowledge*. Interactions between higher order automatization rules and rules lower in a competence hierarchy are governed by the same 'goal switching' control. The result of such interactions are higher *level* rules – *not* higher *order* rules. In the section on transitions, automatization is shown to close a theoretical loop of sorts, making it possible to explain, predict and even control learning over long periods of time. Studying the latter has been largely the exclusive domain of developmental psychologists but it is equally important to instructional scientists.

Higher order rules also play a crucial role in rule selection (e.g., in studies of 'interest', 'direction' or 'motivation') where more than one solution rule is available. In particular, the control mechanism directs the search to higher order rules, irrespective of whether *no* matching rule is found, or whether *two or more* rules are found. In the latter case, the subject is forced to choose. Rather than introducing a fixed (and usually arbitrary) conflict resolution technique (as is commonly done in expert systems), resolution in SLT is handled in the same manner as all problem solving – via higher order selection rules. Irrespective of the selection rules introduced, the same control mechanism is used. This fact was first noted in the literature in (Scandura, 1973, Chapter 8; compare with Scandura, 1971b).

Higher order automatization rules are also governed by the same principles, and have been shown to account for the kinds of behavior associated with Piaget's developmental stages (Scandura & Scandura, 1980). Analysis in terms of rules and higher order automatization rules provided an efficient

and precise account for transitions from pre-operational to concrete operation stages of development. Until pre-operational rules had been mastered, it was impossible for children to learn what Piaget had defined as concrete operational behavior. Furthermore, the analysis also accounted for the problem of horizontal decalage. Pre-operational behavior was observed where and *only* where the requisite pre-operational behaviors had been mastered.

The latter problem had long defied easy explanation within both Piagetian and behavioral frameworks. Automatization leads to the acquisition of rules at progressively higher levels in hierarchies of behaviorally equivalent rules. Ultimately, the top rule is learned. Elementary operations (procedural knowledge) associated with top level rules are mathematically equivalent to simple relations (declarative knowledge). What was originally procedural knowledge becomes declarative in nature (i.e., capable of serving as a basis for new learning. This is true not only in a formal (i.e., mathematical) sense but also behaviorally. Once automated, declarative knowledge effectively defines new I-O elements. These new elements provide the grist for entirely new kinds of learning (as with Piaget's stages – e.g., see Scandura & Scandura, 1980, p. 127). This point is further elaborated below in the section on transitions.

Assessing behavior potential

Given what a learner knows and his current state of knowledge, the above analysis provides a basis for predicting that learner's behavior on any given problem. However, this is rarely known; and one certainly cannot know everything. One cannot look inside a person's head to determine what is known, as one might a computer program (inside a computer). We can only infer what is there by observing the learner's behavior.

The basic question is how to determine most efficiently what a given learner knows at any given point in time. From a SLT perspective three basic problems have been identified.

1. How to distinguish alternative accounts of the same behavior (e.g., how to determine the 'best fit' between borrowing and equal additions in column subtraction? – see Durnin & Scandura, 1973).
2. How to distinguish known and unknown paths in rules in a given rule hierarchy (e.g., the ability to subtract but not when there are zeros in the top numeral? – see Scandura, 1971b; Scandura & Durnin, 1978a; Durnin & Scandura, 1973; Hilke, Kempf & Scandura, 1977; Scandura & Reulecke, Chapter 10 in Scandura, 1977).
3. How to distinguish the atomicity level of equivalent rules (slices) in a given hierarchy? The problem is how best to distinguish between behaviorally equivalent rules, in which the only difference is in process

atomicity and corresponding structural complexity (with top level or atomic rules corresponding to direct declarative knowledge).

Distinguishing between learned paths and levels in a given abstraction hierarchy is relatively well understood and researched (e.g., Durnin & Scandura, 1973). Particularly relevant to instructional design is making knowledge assessment more efficient by testing at multiple levels in a hierarchy (e.g., Scandura et al., 1986; Scandura, 1987b; Scandura & Scandura, 1987). This approach uses the principle of 'splitting the difference' – selecting those test items at each step that provide the most information about which paths are learned and which are not. For example, if a person is successful on a path in the middle of a hierarchy, then he is almost certainly able to solve tasks associated with 'easier' paths (in which learned components are refined into more elementary components). Conversely, failure would imply failure on all paths that are more difficult.

Although the problem of distinguishing alternative accounts of the same behavior has been addressed in earlier research (e.g., Durnin & Scandura, 1973; Reuleuke & Scandura, Chapter 10 in Scandura, 1977), more explicit theory and basic experimental research is needed. Similarly, although response time has been widely used in psychological experiments to distinguish alternative accounts of behavior, this research might well be put on a firmer theoretical basis if higher order processes responsible for atomicity were given more explicit consideration (e.g., Scandura & Scandura, 1980).

Assessment problems become even more complex when seeking to determine what is learned with respect to sets of rules collectively. The number of possible distinctions becomes much greater and assessment considerably more difficult. A key question is whether one should bother with such situations at all. Theoretically, it should be possible to obtain the same information by assessing higher and lower order rules individually. Implementing this approach, of course, requires constructing suitable test items for given higher order rules. The feasibility of doing so has been demonstrated in laboratory experiments (e.g., Scandura et al., 1971; Scandura, 1974).

Transitions

The above analysis (under Cognition) provides a basis for refining answers to basic questions, "Why is it that some people can solve problems that others cannot? And, how is it that initially naïve learners acquire new competence – and gradually come to acquire the mastery associated with experts?" (Scandura, 1981, p. 139). The SLT control mechanism, together with requisite rules, was shown to provide a uniform basis for explaining transitions from naïve problem solver to neophyte. The same mechanism, together with higher order automatization rules, was also shown to account for mastery. Once one

rule in a competence hierarchy has been learned, automatization results in the acquisition of higher level rules – ultimately leading to top-level atomic operations with fully elaborated data structures.

For illustrative purposes, consider the task of constructing a statement expressing one's limited ability to speak German. The basic problem (given → goal) can be represented as:

[Idea: Know little German] → <?Proper descriptive phrase>

The naïve learner approaches the problem with a knowledge base consisting of rules for expressing ideas associated with terms like 'ich' [I], 'Deutsch' [German], 'ein wenig' [a little], 'sprechen' [to speak], 'kann' [can], 'bin' [am] and 'nur' [only]. In order to construct an appropriate response, the learner would also need some equivalent of a higher order rule to the effect, 'put words in the order: subject, initial verb, adjectives and objects, other verbs'.

The neophyte's knowledge base might directly include a statement to the effect "Ich kann nur ein wenig Deutsch sprechen" [I can only a little German speak.] More precisely, this knowledge is represented by a rule for constructing such a statement.

It is almost an oxymoron to refer to a master's knowledge base in this context, so we assume the master is a German teacher. The master's knowledge base is likely to include a wide variety of phrases with very similar meanings: "Ich kann nur ein wenig Deutsch sprechen", "Ich bin im Deutschen ein Anfanger" [I am in German a beginner], and so on. The master's data structures are far more complex, representing a variety of (sub-categories of) meanings and corresponding expressions, enabling the expert to express fine distinctions unavailable to the neophyte.

More generally, transitions from naïve to neophyte status are characterized by the use of higher order rules to construct new rules. Once learned, a new rule allows the neophyte to solve similar problems in a systematic manner. Transitions from neophyte to master status involve application of higher order automatization rules. These higher order rules transition learning from one level in a rule hierarchy to another. They generate higher level (behaviorally equivalent) rules with simpler operations and correspondingly more complex parameters (data structures).

Application of SLT to Piagetian research (Scandura & Scandura, 1980) exposed more global transitions where mastery in one domain made it possible to learn qualitatively different domains. What constitute rules to be learned in one domain effectively become – once mastered – input and output elements for defining entirely new domains. Mastered rules in one domain provide elements for the next.

For example, it is often a painstaking process for young children to learn to construct Arabic numerals, 1, 2, 3, 4, 5, . . . by combining previously learned straight and curved line segments in precise patterns. Gradually, however, the process becomes increasingly easier, faster and more reliable. Ultimately, writing numerals becomes fully automatic, taking only the time required by the physical actions themselves.

Once mastery has been achieved, these former operations effectively become declarative (elemental in nature). Writing numerals can now serve as input and output elements for defining basically new kinds of problems. Teaching a young child how to perform column subtraction, for example, would be meaningless unless the child first knew how to write numerals (or perhaps to type with the aid of a computer). Similarly, it is hard to imagine a person learning to write poems, for example, without having first mastered a reasonable vocabulary.

Scandura and Scandura (1980) found that transition from Piaget's pre-operational stage of development to his stage of concrete operations was possible only after certain operations had been fully mastered (i.e., automated). True number conservation, for example, was impossible until Ss had mastered one-to-one mappings. Until then, number conservation problems were effectively meaningless. According to SLT these principles are universally applicable, and play an essential role in all such transitions. Mastering one domain provides a new foundation on which subsequent learning may be based.

Teaching and learning

This section returns briefly to Figure 1, and particularly to what is meant by 'diagnostic and tutorial expertise'. In oversimplified form, tutorial expertise means asking the right questions and presenting the right information at the right time relative to what the learner knows (Scandura, 1964b, 1987a; Scandura & Scandura, 1987). Methods used to teach various kinds of content derive from a common set of SLT principles. Optimal instructional techniques depend as much on the learner's state (of knowledge) as on the kind of content (e.g., conceptual, procedural, problem solving).

Assuming input and output prerequisites, and well-defined competence, for example, optimized assessment involves choosing test items that maximize the amount of information that can be gained about the learner's state (of knowledge). Initially, the tutor has no idea of what the learner does and does not know. All paths in all rules in the target competence hierarchy are question marks. At each stage of learning, the tutor should choose test items providing the most information. Initially, this item will be associated with some path in a rule located midway in the hierarchy.

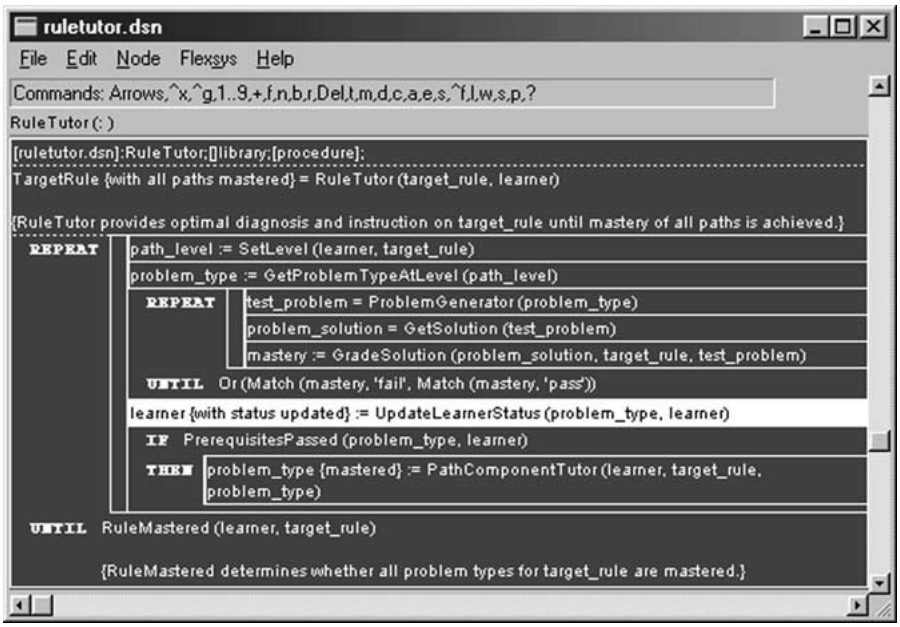

Figure 3. The General Purpose RuleTutor represented as a Flexform in Softbuilder.

Success here implies success on all subsumed paths of rules lower in the hierarchy. Conversely, failure implies failure for all super-ordinate (encompassing) paths in rules higher in the hierarchy. Irrespective of success or failure, maximum information is attained about what the learner knows.

Maximal information also is attained about what the learner needs to know in order to overcome any inadequacies. Instruction should optimally be associated with a path in the hierarchy for which all prerequisite components have been learned. The process of 'splitting the difference' in testing (to obtain maximum new information) and providing just that information needed (to build on what is already known) continues until all paths have been learned. This optimized process for testing well-defined (procedural) content, herein referred to as a General Purpose Tutor (GPT), is represented as a Flexform in Figure 3 (Scandura et al., 1986; Scandura, 1987a; Scandura & Scandura, 1987).

This GPT is entirely independent of the particular rule to be taught. Given a suitable representation of any rule, the tutor can select those test items at each stage of the process, which provide optimal amounts of information about the learner's state. It can also identify exactly the information that the learner needs most. Perfect memory and rapid processing speeds would enable an automated tutor to do this more efficiently, and with more reliability than the best human teacher. The information that must be referenced with

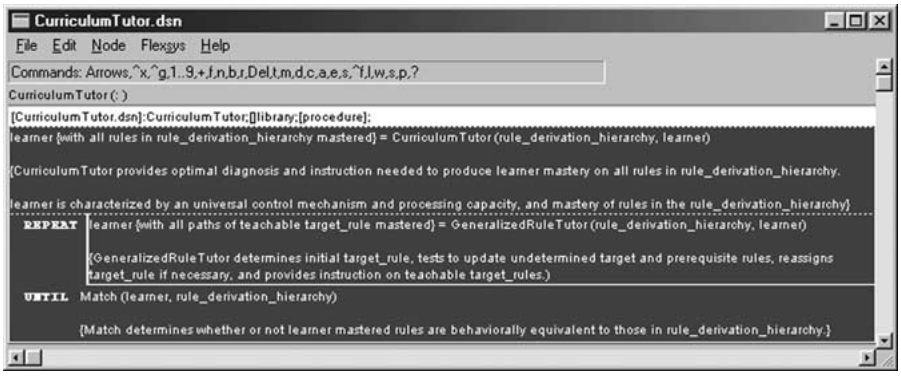

Figure 4. The general purpose curriculum tutor represented as a Flexform in softbuilder.

complex content at each stage of the process is likely to go far beyond what any human might reasonably be expected to keep in mind.

These same principles may be applied in teaching problem solving skills. Higher order rules may be taught in the same manner as other rules. Although discovery learning is often an assumed prerequisite to higher order knowledge, research shows that what is learned depends far more on what is taught than on the way it is taught (Scandura, 1964a,b). The main determinate is the degree to which the target content has been identified and represented in operational terms (as rules). Higher order rules, for example, can often be taught by exposition more efficiently than by discovery (Roughead & Scandura, 1968). The only caveat is that the higher order rules in question must be specified with sufficient precision to enable (direct) instruction. To date, little attention has been given to isolating requisite higher order rules.

What seems to matter most is what a learner knows when information is presented. If given too soon, information goes unused, and can even be misleading. Given too late and the learner can become discouraged, disoriented and/or otherwise lose interest (Scandura, 1964b; Scandura et al., 1969b). Although higher order rules have rarely been taught directly in problem solving instruction, empirical research suggests to the extent the higher rules can be precisely identified that this would be a very efficient way to proceed.

Problem solving behavior is commonly taught by example plus practice.[7] Optimized instructional decision making becomes considerably more complex in this context. Learners typically use various combinations of higher and lower order rules, and it is very difficult both to choose appropriate test problems and to determine the most helpful information.

The GPT design is extended in Figure 4 to accommodate arbitrarily complex (e.g., problem solving) domains. It is designed to accommodate

the learner's use of various combinations of higher and lower order rules in attacking test problems. Assessing higher order knowledge in this context depends on pre-specification of requisite higher order rules.

As shown in Figure 4 (Content), Structural Analysis (SA) can be applied to ill-defined as well as well-defined content domains. In principle, SA can also be applied dynamically as unanticipated issues arise during the course of instruction. A learner might, for example, ask or answer a question that adds a new dimension to any given SA. The tutor has the option of extending the domain of discourse by dynamically extending the content domain and SA thereon. The ability to extend the domain of discourse dynamically during instruction is something that separates superior teachers from others. It is not clear, however, to what extent such abilities can be automated.

The time-honored way to assure that a rule is mastered is practice. Research (as well as common observation) shows that repeating a learned solution results in faster surer responses. The current analysis suggests that more might be done by way of pinpointing higher order automatization rules, and teaching them to help learner's automate previously learned rules. Although supporting research is limited (Scandura & Scandura, 1980), SLT suggests that automatization can be made more efficient by identifying and explicitly teaching automatization techniques. The idea is to identify higher order rules that assist learners to transition from rules lower in a set of competence hierarchies to rules at higher levels. (Note: The distinction between *higher order* and *higher level* is crucial. Any given higher order automatization rule may effect transitions to higher levels in an indefinitely large number of rule hierarchies.)

Technology

Implementing the kinds of tutorial systems described above would be relatively difficult with currently available technologies. Although major advances have been made in recent years in component based software and object oriented programming, for example, current methods make them too complex and difficult to use by most instructional scientists. Indeed, many working in theoretical computer science tend to avoid programming, precisely because of its idiosyncrasies and complexity – things that are not easily learned without spending inordinate amounts of time and effort.

Scandura.com's Softbuilder system approaches this problem with its abstract syntax tree (AST) based high level design (HLD) language. ASTs are widely used in artificial intelligence because they directly reflect real world data structures and operations (e.g., Scandura, 1987b, 1991, 1992, 1994). The need for and use of such programming constructs as data types, global vari-

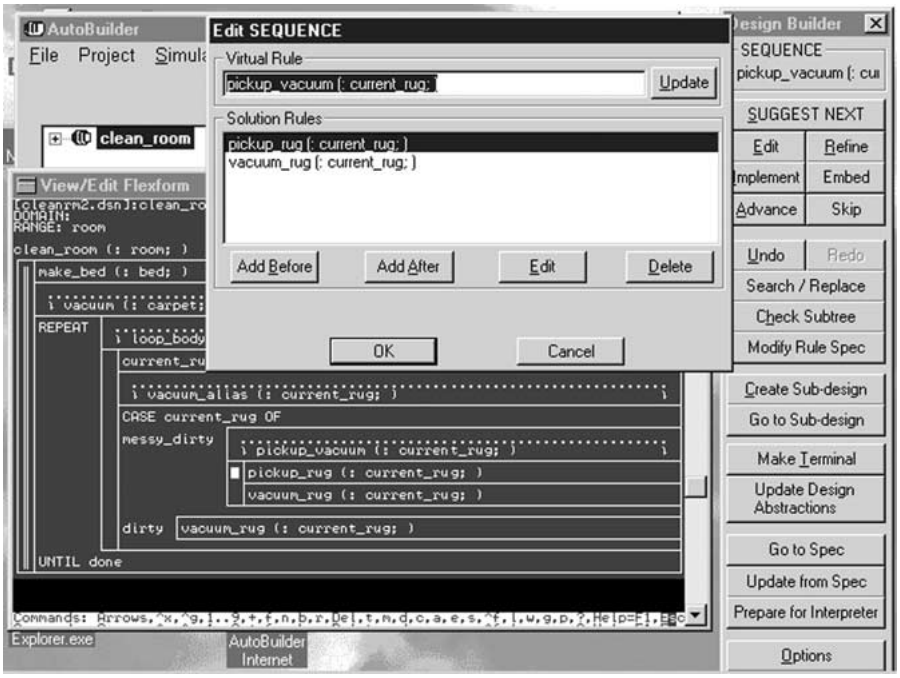

Figure 5. AutoBuilder in the process of refining the virtual component, pickup_vacuum (:current_rug;), into a sequence.

ables, semi-private methods and the like is eliminated entirely. Softbuilder also ensures component interoperability and extensibility (i.e., the ability to add new components as they are needed).

Softbuilder's ability to model the real world on its own terms makes it possible for computer literate instructional scientists (who are not also programmers) to directly participate in building automated tutorial systems.

AutoBuilder research goes further in this direction. In addition to enabling domain experts (as well as programmers) to more efficiently build and maintain complex software systems, AutoBuilder will guarantee that the software will operate exactly as specified. AutoBuilder is based on recently patented processes (Scandura, 2001, in press) and a prototype in which software specifications and designs are represented in semantically meaningful terms at multiple levels of abstraction. These abstraction hierarchies are guaranteed to be internally consistent, with designs guaranteed to be correct with respect to specifications.

A software system built with AutoBuilder is guaranteed to work as a whole if each component in which it is implemented performs as specified. Implementation components can be made arbitrarily small, thereby making it easy

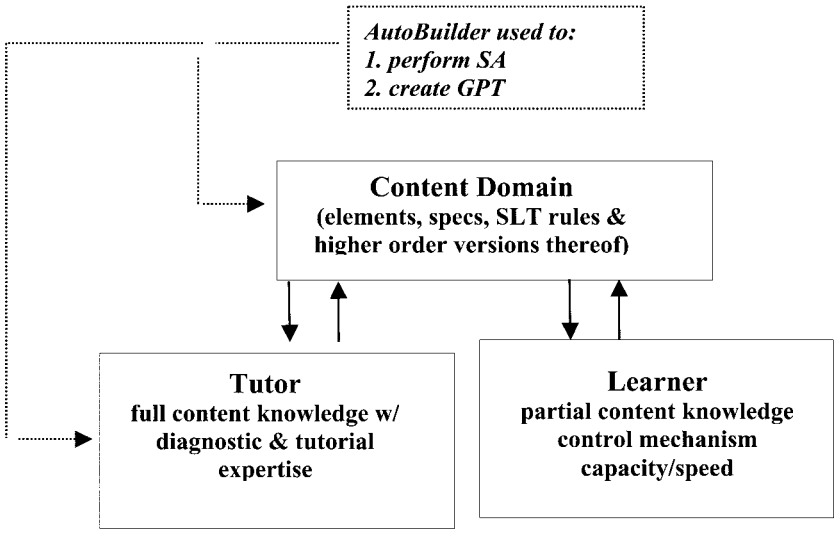

Figure 6. Roles Softbuilder and AutoBuilder will play in construction of the general-purpose tutors described above.

(either by inspection or automated theorem provers) to ensure that they are correct. Figure 5 above shows an internally consistent design hierarchy for cleaning rooms.

Softbuilder and AutoBuilder will play an essential role both in representing the content to be taught and in constructing general-purpose tutors (see Figure 6). Softbuilder includes specific support for interacting systems – for example, interacting tutors and learners. At the highest level of abstraction an Intelligent Tutor may be represented as a dialog between a tutor and a learner with respect to given content.

dialog (content, tutor, learner)

The tutor and learner are independent agents, which communicate with one another via the content. At the top level of abstraction (in a hierarchy) these 'agents' may be characterized as follows:

learner (tutor_event:; learner_event)

Learner is a fully independent agent reacting to *tutor* created events but independent of *tutor per se*.

tutor (learner_event: learner_model; tutor_event)

An automated *tutor* must know what *learner* knows at each stage of learning in order to know what question to ask and how to update his model

of *learner* based on *learner*'s answers.[8] As discussed above, *content* may be well-defined, ill-defined or dynamic.

Concluding remarks

This paper updates and extends the early SLT literature, with emphasis on providing a more complete and coherent picture of SLT. Many instructional designers, for example, have read only the limited version published in Reigeluth (1983). Although many of the ideas expressed herein are natural extensions of the earlier literature, much of that literature is scattered and otherwise inaccessible. Similarly, while motivation for many of the ideas may be found in the early literature (e.g., Scandura, 1971b, 1973, 1977c; Scandura & Scandura, 1980), other key ideas have been developed only recently (e.g., Scandura, 1997, 2001). Higher order rules have played a central role in SLT since its inception. This paper further shows how SLT offers a unified theory of cognition, making it possible in principle to explain all behavior from birth onward. Higher order knowledge produces new knowledge, which in turn gradually becomes more automated. Automatization, in turn, introduces new I–O elements, allowing the process to repeat at higher levels of learning.

A recent 300 plus page patent (Scandura, 2001, in press) details SA and the universal control mechanism. There are many other areas, of course, where further refinement and extension is needed.

Although earlier research will be suggestive in many cases, more work is particularly needed in knowledge assessment. Distinguishing between learned paths and levels in given abstraction hierarchies seem to be well-understood and researched, although many of the ideas should be used more widely in practice. Making knowledge assessment qualitatively more efficient via integrated testing at multiple levels in a hierarchy seems ripe for broad applications. More explicit theory and basic experimental research is needed in other areas. For example, more explicit methods are needed for distinguishing alternative accounts of the same behavior as well as for distinguishing behaviorally equivalent SLT rules in a hierarchy.

In its current state, SLT can and should be applied far more widely in instructional design. Indeed, major parts of structural analysis have been and are being applied successfully in software engineering, and it is possible to envisage broad areas of SLT application in artificial intelligence and intelligent systems generally. SLT could even motivate formal developments in these areas.[9]

In summary, this paper covers a lot of ground: (a) how previously acquired and/or innate knowledge (including elements and rules) are prerequisite to new learning, (b) how that new knowledge is constructed (by interactions

between lower and higher order rules), (c) how mastery (via higher order automatization rules) transitions procedural into declarative knowledge and (d) how the process repeats at higher developmental levels, with declarative knowledge forming new elements on which still higher level tasks may be defined. This process has the potential of accounting for complex learning from birth onward.

The paper also shows that what a learner does and does not know can be determined by observing behavior, and how the information gained provides a foundation for highly efficient teaching and learning. The paper also touches on the kinds of technology necessary for implementing such theory in constructing automated instructional systems and outlines relationships to software research.

Nonetheless, much was omitted due both to space and time limitations. More explicit reference to the previous literature would have been helpful to serious scholars in the area, and certainly more attention could have been given to examples. Many examples have been detailed in the earlier literature but some would clearly benefit from reformulation in modern terms. The current SLT formulation, for example, is more uniformly couched in terms of abstract syntax trees, which are used to represent both data elements and operations (and in turn rules, plans and problems).

Despite these limitations, the author hopes the reader will come away with some sense of the exciting new possibilities SLT offers for fundamental research in both cognition and instruction – and more important that some readers will be motivated to apply and/or extend SLT in the future.

Notes

1. This article is based on a talk first given at an invitation conference on *Epistemology, Psychology of Learning and Implications for Learning and Instruction* at the University of Twente on March 31, 1999 while the author was a Fulbright Professor at Koblenz and Dresden Universities, Germany.
2. The term ‘structural’ in the title refers to the SLT’s original emphasis on complex structured subject matters such as mathematics.
3. Space limitations precluded inclusion of a summary overview of relevant research in structural learning by the author and his collaborators, along with a theoretical discussion of relationships with formal work in logic programming and artificial intelligence. The interested reader is referred to http://www.scandura.com, where an unabbreviated copy of this article may be found under “Knowledge Base”.
4. A Patent detailing key aspects of the process was formally approved in November 2000 and is currently in press.
5. Such differences may correlate with gender (due to well-documented hemispheric differences in male and female brains). Although such differences would be of interest in defining sub-domains of problems, SLT puts the emphasis on individual behavior as opposed to group averages.

6. Concept attainment, for example, provides one, perhaps less than obvious application of this idea. Bruner et al. (1956) showed how concepts could systematically be attained by applying a uniform strategy to a given a set of instances (examples) of the concept. Scandura (1964a) extended this idea to more complex kinds of concept learning based on abstract card tasks.
7. Many mathematicians believe that the only way to become a good problem solver is to solve a lot of problems.
8. The aforementioned analysis of tutor-learner interactions can be generalized to any set of interacting intelligent agents. Interactions between interacting groups of individuals might be formulated using the same basic machinery. For example, the soccer players in recent research by Furbach, Stolzenburg, Baumgartner, Obst et al. interact at multiple levels and, hence, might be conceptualized in this manner.
9. See Appendix B in "Structural Learning Theory" under knowledge base at http://www.scandura.com.

References

Bruner, J.S., Goodnow, J.J. & Austin, G.A. (1956). *A Study of Thinking*. New York: Wiley.

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

Hilke, R., Kempf, W.F. & Scandura, J.M. (1977). Deterministic and probabilistic theorizing in structural learning. In H. Spada & W.F. Kempf, eds, *Structural Models of Thinking and Learning*. Bern: Huber.

Miller, G.A (1956). The magic number seven, plus or minus two. *Psychological Review* 63: 81–97.

Polya, G. (1962). *Mathematical Discovery*. New York.

Roughead, W.G. & Scandura, J.M. (1968). "What is learned" in mathematical discovery. *Journal of Educational Psychology* 59: 283–298.

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

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

Scandura, J.M. (1968a). Research in psycho-mathematics. *The Mathematics Teacher* 61: 581–591 (Reprinted in *The Mathematics Education*, 1968, 3).

Scandura, J.M. (1968b). New directions for theory and research on rule learning: I. A set-function language. *Acta Psychologica* 28: 301–321.

Scandura, J.M. (1969a). New directions for theory and research on rule learning: II. Empirical Research. *Acta Psychologica* 29: 101–133.

Scandura, J.M. (1969b). A theory of mathematical knowledge: Can rules account for creative behavior? *Journal of Research in Mathematics Education* 2: 183–196.

Scandura, J.M. (1969c). New directions for theory and research on rule learning: III. Analyses and theoretical direction. *Acta Psychologica* 29: 205–227.

Scandura, J.M. (1970). The role of rules in behavior: Toward an operational definition of what (rule) is learned. *Psychological Review* 77: 516–533.

Scandura, J.M. (1971). Deterministic theorizing in structural learning: Three levels of empiricism. *Journal of Structural Learning* 3: 21–53.

Scandura, J.M. (1972). What is a rule? *Journal of Educational Psychology* 63: 179–185.
Scandura, J.M. (1973). *Structural Learning I: Theory and Research.* London/New York: Gordon & Breach Science Publishers.
Scandura, J.M. (1974). The role of higher-order rules in problem solving. *Journal of Experimental Psychology* 120: 984–991.
Scandura, J.M. (1977a). A deterministic theory of teaching and learning. In H. Spada & W.F. Kempf, eds, *Structural Models of Thinking and Learning*. Bern: Huber.
Scandura, J.M. (1977b). A deterministic approach to research in instructional science. *Educational Psychologist* 12: 118–127.
Scandura, J.M. (1977c). *Problem Solving: a Structural / Process Approach with Instructional Implications*. New York: Academic Press.
Scandura, J.M. (1978a). A. Discussion of selected issues relevant to structural learning. B. Comments on "Who is the learner". In J.M. Scandura & C.J. Brainerd, eds, *Structural/Process Theories of Complex Human Behavior*. Leiden, The Netherlands: Sijthoff.
Scandura, J.M. (1978b). Structural approach to empirical testing: A contrast with normative methods in cognitive psychology. *Journal of Structural Learning* 6: 89–96.
Scandura, J.M. (1981). Problem solving in schools and beyond: Transitions from the naive to the neophyte to the master. *Educational Psychologist* 16: 139–150.
Scandura, J.M. (1982). Structural (cognitive task) analysis: A method for analyzing content. Part 1: Background and empirical research. *Journal of Structural Learning* 7: 101–114.
Scandura, J.M. (1984a). Structural learning theory. In C.M. Reigeluth, ed., *Instructional Design Theories and Models: An Overview of Their Current Status* (pp. 215–245). Hillsdale: Erlbaum.
Scandura, J.M. (1984b). Structural analysis. Part II: Toward precision, objectivity and systematization. *Journal of Structural Learning* 8: 1–28.
Scandura, J.M. (1987a). A structured approach to intelligent tutoring. In D.H. Jonassen, ed., *Instructional Design for Microcomputer Software* (Chapter 14, pp. 347–379). Hillsdale, N.J.: Erlbaum.
Scandura, J.M. (1987b). A cognitive approach to software development: The PRODOC environment and associated methodology. *Journal of Pascal, Ada & Modula-2* 6: 10–25.
Scandura, J.M. (1991). A cognitive approach to software development: The PRODOC environment and associated methodology. In D. Partridge, ed., *Artificial Intelligence and Software Engineering* (Chapter 5, pp. 115–166). Norwood, NJ: Ablex Publisher.
Scandura, J.M. (1992). Cognitive technology and the PRODOC re/NuSys Workbench™: a technical overview. *Journal of Structural Learning and Intelligent Systems* 11: 89–126.
Scandura, J.M. (1994) Automating renewal and conversion of legacy code into ada: A cognitive approach. *IEEE Computer* April: 55–61.
Scandura, J.M. (1997). Cognitive analysis, design and programming: next generation OO paradigm. *Journal of Structural Learning and Intelligent Systems* 13(1): 25–52.
Scandura, J.M. (2001, in press). U.S. Patent (Formally Approved November, 2000), *Automated Methods for Building and Maintaining Software Based on Intuitive (Cognitive) and Efficient Methods for Verifying that Systems are Internally Consistent and Correct Relative to their Specifications*.
Scandura, J.M., Durnin, J.H., Ehrenpreis, W., et al. (1971). *An Algorithmic Approach to Mathematics: Concrete Behavioral Foundations*. New York: Harper & Row.
Scandura, J.M., Durnin, J.H. & Wulfeck, W.H., II. (1974). Higher-order rule characterization of heuristics for compass and straight-edge constructions in geometry. *Artificial Intelligence* 5: 149–183.

Scandura, J.M. & Releucke, W. (1977). Some Statistical concerns in structural Learning. In J.M. Scandura, ed., *Problem Solving: A Structural / Process Approach with Instructional Implications* (Chapter 10). New York: Academic Press.

Scandura, J.M. & Brainerd, C.J., eds. (1978a). *Structural/Process Models of Complex Human Behavior*. Alphen aan den Rijn, The Netherlands: Sijthoff & Noordhoff.

Scandura, J.M. & Durnin J.H. (1978b). Assessing behavior potential: Adequacy of basic theoretical assumptions. *Journal of Structural Learning* 6: 3–47.

Scandura, J.M. & Scandura, A.B. (1980). *Structural Learning and Concrete Operations: An Approach to Piagetian Conservation.* New York: Praeger Science Publisher.

Scandura, J.M., Stone, D.C. & Scandura, A.B. (1986). An intelligent RuleTutor CBI system for diagnostic testing and instruction. *Journal of Structural Learning* 9: 15–61.

Scandura, J.M. & Scandura, A.M. (1987). The intelligent RuleTutor: a structured approach to intelligent tutoring – Phase II. *Journal of Structural Learning* 9: 195–259.

Scandura, J.M. & Scandura, A.B. (1999). Improving RAM in large software system development and maintenance. *Journal of Structural Learning and Intelligent Systems* 13 (3–4): 227–294.

Voorhies, D. & Scandura, J.M. (1977). Determination of Memory Load in Information Processing. In J.M. Scandura, ed., *Problem Solving: a Structural / Process Approach with Instructional Implications* (Chapter 7). New York: Academic Press.

Wulfeck, W.H. & Scandura, J.M. (1977). Theory of adaptive instruction with application to sequencing in teaching problem solving. In J.M. Scandura, ed., *Problem Solving: a Structural / Process Approach with Instructional Implications* (Chapter 14). New York: Academic Press.