Structural (Cognitive Task)
Analysis: A Method for Analyzing Content
PART II: TOWARD PRECISION, OBJECTIVITY AND SYSTEMATIZATION*
JOSEPH M. SCANDURA
University of Pennsylvania

D. NEED FOR GREATER PRECISION AND OBJECTIVITY

As described in Part I (Background and Empirical Research, Scandura, 1982),
the major weakness of structural analysis lies in the high level of description. It
is one thing, for example, to talk about identifying a sample of representative
problems, or rules for solving these problems. It is quite another to devise a
precise representation of the problems in the sample, or of the solution rules,
which adequately captures the essentials. Specifying a method of analysis which
guarantees representational precision lies at an even higher level of difficulty.

Thus, for example, while high level description may provide adequate
guidance for sophisticated analysts who are well versed in the method (and
underlying theory), in general it will not be adequate for less experienced
analysts and, certainly, will not provide the level of procedural detail that would
be needed if one were to automate the process—say with computer assistance.

This limitation is inherent in most of the steps identified:
(1) Exactly what do we mean by a representative sample of problems? How
do we find one? How are the sample problems to be represented?
(2) Just what is a solution rule? And how can we generate and formally
represent them?
(3) In general, what do we mean by parallels among solution rules? And
exactly how can these be realized as higher-order rules?
(4) Given answers to the first three sets of questions, eliminating redundant
rules would seem to be a largely mechanical, if time consuming, process.
(5) Regarding iterative (repeated) analysis: In what sense are higher-order
parallels and higher-order rules like solution rules? How different?

Further analysis of the above steps (which are described more fully in
Scandura, 1977; Scandura & Scandura, 1980) indicates that steps one and two
* This research was supported by a grant from the Spencer Foundation and is gratefully
acknowledged.
are basic to all of the others. Suppose, for example, that we have identified a representative sample of problems and have constructed a solution rule for solving each of these problems. The crucial insight is that, once the solution rules have been identified, they themselves may be viewed as problems representative of related higher-order problem domains. Specifically, each solution rule may be thought of as the solution of a problem in a higher-order domain, namely a domain consisting of problems whose goals are satisfied by the solution rules. In this sense, step three above actually involves two steps, first, identifying the higher-order problems associated with the identified solution rules, and second, of identifying the higher-order rules necessary for solving them. We shall return below to this part of structural analysis.

In the meantime, it is apparent that an explicit formulation of steps one and two of methods for representing problems and solution rules, would go a long way towards the systematization and objectification of structural analysis. There are two aspects of this goal: the precise way in which problems and solution rules are to be represented, and the method of analysis itself (by which the representations are generated). We consider these aspects in turn.

E. REPRESENTATION OF PROBLEMS AND RULES

Clearly, any description of a problem or rule is a representation. The type of description that is most useful, however, depends very much on the intended use. Thus, for example, the descriptions used in the examples of Part I were generally adequate for illustrating the general points we wanted to make about structural analysis. They also provided a generally adequate basis for designing instructional programs dealing with the associated content.

The precision with which underlying rules may be represented, however, depends directly on the type of representation used. Formal mathematical representations provide the greatest degree of precision and, hence, are to be preferred in demanding applications such as in computerized and/or otherwise automated instruction. Most directly relevant here, formal representation is an essential prerequisite to any serious attempt at automating structural analysis. Systematic methods for devising ambiguous representations (of problems and rules) would be a contradiction in terms.

In structural learning theories (e.g., Scandura, 1971, 1973, 1977, 1980) the concept of “structure” is essential. Specifically, a structure is a “discrete” cognitive construct used to represent any encoding of reality and/or any static, non-operating memory.

The term “discrete” is used advisedly. While I personally believe that the human mind processes information both symbolically (i.e., discretely) and analogically (i.e., continuously), that dual coding (e.g., Paivio, 1978) is an essential ingredient of human mental life, the current analysis is restricted to the symbolic. (For evolving discussion of iconic/analogue processing from a structural learning perspective, the interested reader is referred to Scandura, 1970, pp. 521–26; 1973, pp. 95–111 & 236–254; 1977, pp. 256–28, and especially 1981a.)

Formally speaking, a structure, the discrete, symbolic aspects of any encoded reality or memory, may be represented in terms of: (1) a finite set of symbolic elements corresponding to minimal, psychologically relevant units (e.g., percepts, processes), (2) a finite set of relations defined on the elements, and (3) a finite set of higher-order relations defined on the relations (and/or other higher-order relations). For example, the column subtraction display

\[
\begin{array}{ccc}
432 & - & 129 \\
\end{array}
\]

may be represented as a structure in which the basic elements are “–”, “—”, and the individual digits. The basic relations are the rows (i.e., numbers) on which the operation is defined (e.g., 432) and the columns.

To illustrate the generality of the construct, consider the following less clearly structured display associated with Piagetian number conservation.

\[
\begin{array}{ccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
X & X & X & X & X & X & X & X \\
\end{array}
\]

In this case, the basic elements are the 0’s and X’s, before and after the transformation \( t_0 \) (on the X’s), and the observable transformation \( t_0 \) itself. The relations are the sets of O’s and X’s and the higher-order relations are the pairs of sets before and after the transformation.

The notion of “structure” can be generalized in a natural way to include sets of structures. Specifically, in structural learning theories a “structure schema,” analogous to but formally more definite than what others have called a “frame” (e.g., Minsky, 1975), is simply a structure in which one or more of the elements, relationships and/or higher-order relations has been replaced by a variable with a specified range. Thus, for example, if one were to replace the digits in the above column subtraction structure with variables ranging over the digits 0, 1, 2, . . . , 9 (disallowing 0’s in the third column), the result would be equivalent to the class of three digit column subtraction displays. Similarly, the Piagetian Conservation display might be generalized by replacing the given transformation \( t_0 \) (which “clumps” the X’s) with a variable, allowing a set of different kinds of transformations on the X’s – for example, transformations which translate the
N's linearly to the right (of the O's), which spread the X's apart, etc. (e.g., see Scandura, 1980).

In structural learning theories, problems are simply structure schemas in which one or more of the variables and certain of the elements and/or relations are distinguished. Specifically, the distinguished variables define the problem goal and the distinguished elements/relations, the given of the problem. In the case of column subtraction, for example, the top and bottom numbers (rows) constitute the given and the variable representing the possible numbers under the "______" constitutes the goal. Consequently, each of these would be "distinguished" in a formal representation of the subtraction problem. (Note: It is unimportant how an entity is distinguished. In a formal representation of a problem, the important thing is that the distinguished entities are somehow denoted differently so that they can be distinguished from other entities needed to define the problem.)

Correspondingly, the natural "scope" of a problem, or the associated problem domain, is determined by the range of values that may be ascribed to the "givens" of the problem. In effect, each problem is associated with a corresponding domain (or subdomain) of problems. Roughly speaking, these are the problems that in some sense are viewed as equivalent to one another. As we shall see below, problems belong to a common domain if they may be constructed from one another by varying the problem given only to the extent that the problems are associated with the same solution procedure. Such a domain and procedure, together with the associated range, constitute a rule, see below.

Notice is this regard that we are speaking of domains in two senses: domains associated with particular problems, and initially given a priori-domains. (Notice that the latter, e.g., might range from everyday decision making at one extreme to column subtraction at the other.) Just because a problem is sampled from some initially given domain does not necessarily mean that the domain associated with that problem will be the same as the initially given domain. It may or may not be. Thus, for example, any two numbers, where the top one is greater than or equal to the bottom one, may be constructed from any given subtraction problem by varying the values of the digits and/or the numbers of digits in the given numerals. Hence, it is natural to equate the domain associated with any given subtraction problem with the domain of subtraction problems. This would not be the case, of course, if the initially given domain had been the domain of ARITHMETIC problems. In this latter case, for example, one also would have to change fixed properties—such as the "=" sign.

The term "rule" also plays an essential role in all structural learning theories. As formally defined, a rule is a triple consisting of a domain, or class of structures (a structure schema) to which the rule applies, a range, or class of structures which the rule is expected to generate, and a restricted type of procedure which operates on structures in the rule domain.

To understand the nature of the restrictions imposed on the procedures of rules, let me first describe what I mean by a procedure. Roughly speaking, a procedure or algorithm is a recipe, process, technique, or systematic method for doing something. (The term "algorithm" is often preferred in computer science.) More precisely, according to Knuth (1968), a procedure or algorithm must:

1. always terminate after a finite number of steps,
2. include only definite steps that are precisely defined, with actions that can be carried out rigorously and unambiguously,
3. have an associated (possibly empty) class of inputs, or domain,
4. generate at least one output, and
5. be effective in the sense that all of the operations to be performed must be sufficiently basic that, in principle, they can be done exactly and in finite time by a man using pencil and paper.

Not only any entity that satisfies the above constraints would qualify as a "procedure" in the restricted sense used in structural learning theories. For one thing, the procedures must be structured (e.g., Dijkstra, 1968). Each step in a structured procedure must be decomposable into one of three types of components:

(a) Sequences of steps or operations,
(b) Conditional steps or branching (selection) and
(c) Iteration or looping.

These types are illustrated in two ways in Figures 1a, b and c, where the rectangles represent arbitrary operations (e.g., add $a$ and $b$), the diamonds represent arbitrary "if" conditions (e.g., If the building is over 20' tall, then . . .), and the hexagons (and their's) represent arbitrary "while" conditions which involve iteration (e.g. While there is still further to go . . .)

![Diagrams](image_url)

Figure 1: Diagrams a, b & c

There is no loss of generality in requiring that a procedure be structured because it has been mathematically proven that any procedure/algorithm can be represented using these three types of components. In structural learning
theories, however, we impose an additional type of restriction on the steps of procedures. Specifically, no step may involve recursion — this is, no step may be defined in terms of itself, or for that matter in terms of another step defined in terms of itself. The factorial, \( n! = (n-1)! \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1 \), for example, is frequently defined in programming in terms of itself. Thus, \( n! = n \cdot (n-1)! \) (if \( n > 1 \) and \( 1! = 1 \)). In this case, before \( n! \) can be computed, \((n-1)\) must first be computed. The same is true of \((n-2)!\) and so on until we get to \(2!\) which is equal to \(2 \cdot 1 = 2\). Once we have \(1\), of course, we reverse our direction and determine \(2! = 2 \cdot 1\), then \(3! = 3 \cdot 2!\), then \(4! = 4 \cdot 3!\), etc. Notice in this regard that the step involving \(n!\) is defined in terms of the step involving \((n-1)!\) which in turn depends on the step involving \((n-2)!\), etc. It is this dependence of one step on other steps that is explicitly disallowed in structural learning theories.

In structural learning theories the role of recursion is taken over by a postulated universal control mechanism which governs the way individual (and independent) rules interact. This "goal switching" control mechanism allows (indeed, under appropriate conditions requires) some rules (i.e., higher-order rules) to operate on other rules, thereby creating new rules. In the case of \(n\), for example, the role of recursion is taken over by a higher-order rule which serves to construct product rules like \(3! = 3 \cdot 2!\) from known rules like multiplication and \(2!\). Repeated application of such a higher-order rule would sooner or later construct the rule \((n-1)!\) (now well defined as \((n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1\)) and ultimately \(n!\) itself. For further discussion of the respective roles of rules and the control mechanism see Scandura (e.g., 1971, 1973, 1977, 1980).

F. TOWARD OBJECTIVITY AND SYSTEMATIZATION IN STRUCTURAL ANALYSIS

The ESSENTIAL first steps in structural analysis are to identify: (1) the educational goals — the domain of tasks that the learner should be able to solve after instruction and (2) prototypic cognitive processes or rules — those that the learner must learn if he is to successfully perform tasks associated with the educational goals.

As noted previously, systematizing the processes, by which problem domains and procedures (for solving problems in these domains) are specified, would go a long way toward solving the more general problem of objectifying structural analysis.

In the case of problem domains, recall that they may range from extremely broad to very narrow — from such broad domains as reading and writing, or even everyday decision making, at one extreme, to such relatively simple domains as column subtraction and adding "ing" to verbs at the other.

The basic difference between these kinds of domains involves the numbers and varieties of problems they include — and specifically, whether or not the domains associated with given problems are identical to the initially given, a priori domains. (Later on, we shall further distinguish between a priori domains, which ARE the simple set unions of domains associated with finite numbers of problems in the domains, and a priori domains which are NOT. The latter type of a priori domain includes additional problems over and above those associated with the domains of ANY finite sample of problems in a a priori domain.)

Insofar as structural analysis is concerned, a priori domains are subjective constructs of the analyst or serve generally to convey the domain of concern to others. The only domains which play an objective role in structural analysis are those associated with particular representative problems in the domain. (As we shall see, more global domains are formally characterized by finite sets of particular domains, including domains associated with higher-order problems.)

Consequently, the primary objective of systematization involves the process by which the domains associated with specific sampled problems are constructed.

For this purpose it is convenient to consider simple a priori problem domains, which are equivalent to the explicitly represented domains associated with arbitrary problems sampled from corresponding a priori domains. Simple column subtraction, for example, has this property. In this case, the entire domain of subtraction problems can be constructed from any given problem by substituting various digits for the given ones and/or changing the number of digits in the given numbers. Hence, no matter which subtraction problem or problems might be selected as representatives of this domain, the associated problem domains will be the same. What are commonly referred to as structured domains frequently are of this type. (As our work shows, e.g. Scandura, 1977, Chapters 3 and 4, one must be careful not to equate simple "structured" domains with mathematical domains.)

Our initial steps toward further objectivity and systematization, then, refer to a priori problem domains of this type. Having identified a given problem's domain, the analyst next must identify one or more prototypic PROCEDURES for solving the problems in this domain. These procedures must be represented in terms of operations and branching and/or looping conditions which operate only on problem characteristics that are assumed encodable by ALL students in the target population. The operations and conditions of the decisions also must be ATOMIC in the sense that, for each student in the target population, the ability to correctly use a condition/operation once is indicative of uniform success and, conversely, for failure. For example, in column subtraction, borrowing correctly in one case might be assumed to indicate availability of an atomic borrowing rule. Notice that knowing this rule includes both the operations involved and, for example, the decision condition, "Top number less than bottom number?"

The flow diagram in Figure 2 depicts the procedural portion of a rule for per-
forming column subtraction based on equal additions. In this rule, it is implicitly assumed that each operation acts only on digits, rows and columns—consequentbly, the previously referred to need to assume certain minimal encoding/decoding abilities. According to structural learning principles, only to the extent that these assumptions are met, this subtraction rule provide a useful and operationally precise basis for designing efficient and effective instructional strategies.

In view of all these requirements on the way problems and rules are to be represented, one might wonder whether, or to what extent, some such level of representation can be found. Fortunately, the answer is straightforward and unequivocal. For any given student population and any problem domain for which there exists some solution rule, then it is possible to find a representation which satisfies the above requirements (e.g., see Suppes, 1976; Scandura, 1970, 1976). This is accomplished by repeatedly refining (breaking down) some given rule into finer and finer steps until each step (condition/operation pair) is either atomic or universally available to every student in the target population. (While some such level of representation is always possible in principle, in practice the analyst may be satisfied with a less demanding level of precision.)

**G. KINDS OF HUMAN JUDGMENT REQUIRED**

Specifying problem domains and solution procedures involves several kinds of human capacity. First, the analyst must be thoroughly familiar with the content (e.g., a priori problem domain) and with methods for solving problems in the domain, with what constitute representative problems in the domain and their structure and what constitute desirable or prototypic methods of solution (i.e., solution rules). The analyst also must be aware of the minimal capabilities of the subject (student) population, as well as of the general nature of these capabilities as they pertain to the problem domain.

In addition, the analyst must be able to verbalize or otherwise adequately represent his content and population knowledge. At a strictly formal level, any of a wide variety of structural/procedural formalisms, computer programs, etc. might appear to suffice for these purposes. However, for reasons mentioned above and detailed in Scandura (1977, 1980, 1981), representation must be in terms of rules and the problems on which they operate.

By implication, in addition to being able to construct problems and to carry out appropriate solution rules, the analyst must be able to identify the important features of the problems and the solution rules. The analyst also must be able to determine the fit between important features of the problems and solution rules, and the minimal capabilities that may be assumed available to all students in the target population. These assumptions necessarily pertain to both components and overall structure (of problems) or form (or potentially available solution rules).

The existence of numerous structural analyses indicates that content can be analyzed in terms of problems and rules as defined. The fact that many structural analyses have been verified empirically further indicates that at least some analyses are able to make appropriate judgments with respect to subject populations. As shown in previous research (e.g., Scandura, 1977; Scandura & Durnin, 1978), the prototypic rules identified via structural analysis have been shown to provide reliable bases for operationally defining the knowledge, or behavior potential, of individual students.

**H. SPECIFYING PROBLEM DOMAINS**

Given the rule as our basic representational formalism, then, the major theoretical task remaining is that of explicating the base of knowledge needed for Structural Analysis. Preferably this knowledge base should be rendered in procedural form—a procedure which can be carried out by neophyte analysts sharing certain presupposed requisites.

Let us first consider a method for formally specifying the domain of a given problem. While the method is perfectly general, it is convenient and perhaps more meaningful to illustrate with respect to simple subtraction.

Informally speaking, the educational goal in the case of subtraction refers to the class of column subtraction problems displayed on paper. Each problem in this domain involves finding the difference and writing the corresponding numeral beneath the "-".

In general terms, the first phase of structural analysis involves selecting a sample problem associated with the goal in question. For instructional purposes,
representative problems may be viewed informally as problems that the
"teacher" feels best represent collectively the kinds of problems in the domain
that a knowledgeable person should be able to solve. In the case of simple sub-
traction, a sample might include the problems:

\[
\begin{array}{cccc}
9 & 879 & 432 & 402 \\
-5 & -325 & -129 & -129 \\
? & ??? & ??? & ??? \\
\end{array}
\]

Specifying the \textit{a priori} problem domain, at a minimum, involves determining
the "natural" domain associated with each sample problem. With the problem as
a base, the analyst asks which features (properties, relations, operations) of the
problem of the same type (i.e., which can be solved via the same intuitively
known rule). Proceeding in this manner, the analyst replaces each such feature
with a variable, together with its range of permissible values. The resulting prob-
lem \textit{schema} denotes a class of problems, specifically the domain associated with
the problem in question.

According to Scandura (1981), accomplishing the above involves two identifi-
able sub-steps, which we illustrate in the case of subtraction:

(a) Assumptions must be made regarding the MINIMAL ENCODING AND
DECODING CAPABILITIES of the students in the target population. In the
case of second graders, for example, the teacher/analyst would normally assume
that all students are able to distinguish "the minus sign", the individual digits 0,
1, \ldots, 9, the columns, the rows, and that all are able to write the individual
digits in desired locations. Consequently, the remainder of any analysis will be
adequate just to the extent that these assumptions reflect minimal student
capability with respect to any given target population.

(b) The analyst also must decide the SCOPE of each of the representative prob-
lems. This scope effectively defines the domains of the rule associated with the
prototype. The problem

\[
\begin{array}{c}
432 \\
-129 \\
?? \\
\end{array}
\]

for example, might be held prototypic of the entire class of column subtraction
problems, namely those formed by varying the individual digits 0, 1, \ldots, 9 and/or
the number of columns. More specifically, the domain associated with this
problem can be constructed from the problem by replacing each of the digits
and the number of columns by a variable ranging over the digits 0, 1, \ldots, 9 and
the number of columns \( n = 1, 2, \ldots \), respectively. In the present case, notice

that every column subtraction problem is prototypic of this same domain. Con-
sequently, in the present case, it is reasonable to assume that there is a single,
well-defined domain, the domain of column subtraction problems.

I. SPECIFYING SOLUTION PROCEDURES

Consider next the task of detailing the process by which solution procedures are
identified. In this regard, recall that the procedures in structural learning-type
rules must be structured. They must be represented exclusively in terms of
sequences of operations, decisions, and iterations. In addition, such procedures
explicitly disallow the use of recursion. Detailing structural analysis in this case
becomes more feasible because representing procedures in structured form im-
poses more constraints on the process (than would be the case were arbitrary
procedures allowed).

There are two quite different ways of approaching this problem: one called
"top-down analysis" and the other "step-by-step analysis."

J. TOP-DOWN ANALYSIS

The procedures at each stage of top-down analysis must necessarily be structured.
In a "top-down" approach to structural analysis, the first step is to represent the
procedure in question as a single unitary operation. Any operation, then, may be
refined by breaking it down into sequence of operations, a decision which selects
between two possible operations, or an iteration of an operation. Each operation
is such a refinement can be further refined or broken down.

For example, consider the case of column subtraction, where the solution
rule in question is the commonly taught method of borrowing (or regrouping).
The first step in refining the subtraction operation is to observe that subtraction
is performed by column, beginning on the right. In effect, at the second
level, the subtraction procedure can be represented by a sequence of "subtract
the column" operations, each one involving a different column. (Notice,
however, that the actual process of subtracting a given column may require
reference to digits not in that column, as in "borrowing.")

If column by column subtraction is performed with different problems, of
course, particularly with problems involving different numbers of columns, the
number of operations involved will vary. One always starts subtracting with the
rightmost column. What varies is the number of subsequent columns that must
be subtracted.

This type of variation, where there may be from 0 to \( N-1 \) (additional) applica-
tions (where \( N \) is the number of columns) of the operation "subtract the
column," calls for an ITERATION. Specifically, it is an iteration defined on that
characteristic of the problem which causes the variations observed, namely the
number of columns after the first. The steps involved at the first and second levels of refinement are illustrated below.

```
Level 1
  Subtract

Level 2
  Go to Rightmost Column
  Subtract 1st Column
  While More Columns
    Go to Next Column
    Subtract Column
```

As indicated, the first thing the analyst observes is that the subtraction operation may be performed by starting on the right and subtracting column by column. This involves a sequence of three types of operations: first going to, second subtracting the rightmost column, third subtracting columns individually up through the last column. Since the third type is variable, that is, may involve any number of steps (one for each column), it would not be allowed as such in a structured program or procedure. Therefore, at level two, the variable part of the procedure is replaced by an iteration on the number of columns, as shown in the level two figure.

The subtract column operation may be further refined. In particular, what one does in subtracting a column depends on whether the top number is greater than or equal to the bottom number. If the bottom number is larger, the subtraction procedure calls for "borrowing" from columns to the left. Once a column has been subtracted, one proceeds to the next column in the same way, irrespective of whether borrowing has been involved or not. This type of situation calls for a branching condition, sometimes called an alternation, and may be represented as shown in the Level 3 figure given below.

```
Level 3
  Go to Rightmost Column
  Subtr. 1st Column
  While More Columns
    Next Column
    Top > Bottom
     + Subtract
    Borrow Subtract
```

There is more to this particular solution procedure, in the sense that how one actually borrows depends on whether the next digit over in the top row is or is not a zero, but these details can be ignored for the moment.

The above method constitutes a small step toward the goal of systematizing structural analysis. Rather than having to construct solution procedures entirely de novo, the modified form of top-down analysis proposed makes it possible to do this in an incremental fashion. Specifically, top-down analysis has the advantage of specifying the "flow of control" as well as of allowing one to specify components in the solution procedure at any given level of detail. In and of itself, of course, top-down analysis gives little indication as to what is the appropriate level of analysis in any particular situation. These judgments must be made by the analyst.

More important, top-down analysis requires a good deal of intuitive judgment in proceeding from one level of analysis to the next. Certainly more than one might want in a totally objective and systematic method. While each step involved seems reasonable insofar as it goes, considerably more detail is required as to how to proceed from level to level.

In this regard, notice that the type of step (i.e., sequential operation, selection, iteration) called for in a solution procedure depends on the nature of the sequences of actual operations and decisions actually performed in solving particular problems. These solution sequences sometimes are called "computations" in computer science. For example, the number of times the "subtract column" operation must be applied clearly depends on the number of columns in a problem. But this type of variation over problems is unnecessarily represented as an iteration in the subtraction procedure itself (which applies to all problems).

K. STEP-BY-STEP ANALYSIS

The relationship between solution procedures and sequences of solution steps involved in solving particular problems provides the basis for a second approach to procedural specification. This approach, which I call "step-by-step analysis", involves specifying solution procedures by starting with the specific problem's and sequences of solution steps (states). The rationale is that whereas subject matter experts are not always able to devise general solution procedures, they are able to specify problems and their solutions, including the various intermediate states involved in solving particular problems. Why not let them start with the specifics?

Using this approach, one starts with a specific problem and constructs a solution sequence, step by step (i.e., state by state), exactly as the analyst would have the students solve the problem. In so doing the analyst is required to specify the critical features of the solution state at each step (i.e., those features which determine the next step). Then, for each successive pair of states, the analyst
specifies the atomic rule deemed to govern the transition from the first state (of each pair) to the second. (In practice, it normally is sufficient to specify only the domain conditions and the operation (atomic procedure) of the atomic rules because the ranges of the atomic rules normally are defined to be the image sets of the domains under the operation — i.e., the set one gets in applying the operation to all possible entities in the domain.) For example, in the case of column subtraction, the pair of steps below

\[
\begin{array}{c|c}
\text{Before} & \text{After} \\
xx6x & xx6x \\
xx2x & xx2x \\
\hline \\
x & 4x
\end{array}
\]

corresponds to an atomic facts rule, where the domain is the set of basic subtraction facts (i.e., 0-0, 1-0, 1-1, ..., 18-9) and the atomic operation generates the corresponding differences.

The next step in step-by-step analysis is to determine for each atomic rule in the sequence whether any "extra-domain" conditions must be satisfied in order for the atomic rule to be applied. ("Extra-domain" conditions are conditions which refer to characteristics of problems or solution states that are not normally part of the definition of the domain of the following atomic rule.) In the above example, for instance, the atomic facts rule applies only where the top digit is greater than or equal to the bottom digit.

In complex applications, the most difficult step is analyzing the resulting sequence of extra-domain conditions and atomic rules for the presence of repeating patterns. In this case, each type of pattern reflects one or another of the three basic types of constructs allowed in structured programming. For example, extra-domain conditions correspond to iterations where some sequence of successive operations after the condition (beginning with the one immediately after the condition and with no intervening unclassified extra-domain conditions) generates an output which satisfies that condition. Here, the condition defines an iteration. Otherwise, the extra-domain condition defines a decision or selection.

Notice in the above subtraction example that there is an "extra-domain" condition immediately after subtracting digits in a column. Further, since the atomic facts rule never activates (or generates) a new fact (i.e., pair of digits) but rather a difference, the above extra-domain condition (i.e., top digit greater than or equal to bottom digit) is never called upon again and, hence, does not define an iteration. As there are only two possibilities where extra-domain conditions are involved, the latter condition defines a selection.

Generally speaking, any given solution sequence will specify only part of the corresponding solution procedure. However, one may flesh out the first partial solution procedure, determined as above, by comparing it with solution step sequences corresponding to other problems. In general, a finite number of solution sequences will be sufficient for fully specifying a solution procedure.

All in all, this step-by-step approach has much in its favor. By forcing the analyst to solve problems exactly as he would have students solve them (after instruction), by implication choosing a level that can be communicated to all students, the appropriate level of analysis is determined directly. Also surprising is the extent to which the process might be automated. Major portions of the approach would appear to lend themselves quite well to computer implementation, thereby serving to automate a number of otherwise tedious, time-consuming, and/or complicated tasks.

In its simplest forms, however, step-by-step analysis has two major limitations. First, the approach frequently requires extremely complicated analyses which are very demanding of human capability. Although computer implementations would ameliorate this problem, the fact remains that we do not have such an implementation, and that it would require a great deal of effort to develop one. Second, and more fundamentally, step-by-step analysis is sometimes ambiguous in certain respects pertaining to the overall flow of control. This is particularly true where two or more extra-domain conditions correspond to overlapping sequences of atomic rules. In such cases, the solution procedures associated with given sequences of solution steps are not always uniquely determined by the method of analysis.

L. COMBINED TOP-DOWN/STEP-BY-STEP ANALYSIS

In summary, top-down analysis has the advantage of making clear to the user the "flow of control" of the solution procedure. It also allows for arbitrary levels of procedural refinement. Top-down analysis, however, deals with specific problem characteristics only in an informal heuristic manner.

As we have observed informally, there is a close connection between the domain of a problem and the corresponding solution procedure. Informal awareness of either one necessarily influences specification of the other. It would be highly desirable to make such awareness public.

Step-by-step analysis, by way of contrast, deals more directly with particular problems and solution steps. It also provides a potentially mechanical means of translating sequences of solution states into general procedures. But, there are sometimes ambiguities as to overall structure and flow of control. This ambiguity derives from lack of specification of the interrelationships between problem domain and procedural representation.

Clearly, a method of analyzing procedures which combines the advantages of top-down and step-by-step analyses could go a long way toward providing the kind of objective and systematic method we seek, and it is to this possibility that we turn next.
One method is to employ problem and step-by-step analyses at progressive levels of top-down analysis. In this method the problem representation and the solution algorithm are refined in parallel. We illustrate the method in the context of subtraction, while emphasizing general principles.

1. First, the analyst would construct a column subtraction problem (on intuitive grounds). For example:

\[
\begin{array}{c}
437 \\
-275 \\
\end{array}
\]

This problem, then, would be used to specify the corresponding problem domain to a first level of approximation. Toward this end, the analyst would employ the following procedure:

A) Identify the minimum essential characteristics of the problem which make it a problem of the desired type. Specifically, with regard to column subtraction, the "-" sign, the "-----", and the two numerals (437 and 275), where the one on top designates a number that is larger than the bottom one, constitute the "given" of the problem. The unspecified solution numeral (the one that goes under "-----") specifies the problem goal.

B) Identify those input (i.e., "given") characteristics which may vary without changing the essential nature of the problem. By essential nature is meant those qualities of the problem which make it solvable via the, at this point, only intuitively known solution procedure associated with the problem. In the case of a column subtraction problem these characteristics are the two numerals, where the top one is greater than or equal to the bottom one.

C) Replace each such input characteristic with a variable and specify the sets of allowable values of these variables. In the case of subtraction, there are two domain variables: Top number (T), and bottom number (B), such that T > B.

Carrying out Steps A, B and C yields a problem schema which formally represents the domain of the associated rule to a first approximation.

Given the rule domain, the analyst next solves the original problem as specified and uses the resulting states as a basis for specifying the solution procedure. More exactly, the analyst uses the following method:

D) Take the specified values of the input variables of the original problem as inputs and apply step by step the solution procedure, which at this point is known only intuitively. In the case of the above subtraction problem, the analyst just subtracts the two given numbers — i.e., specifies the solution. In doing this, he can refer formally only to the variables (numbers) as wholes. Hence, the result is just a pair of states, the subtraction problem before and the subtraction problem after the difference has been written. (The analyst cannot refer to individual digits, for example, since they do not yet exist from the standpoint of the formal analysis. They are known intuitively to the analyst, but have not yet been represented formally. Hence, the digits cannot be referred to explicitly in formally representing the solution procedure at this level.)

E) Go to the first pair of states. (In this case there is only one pair of states: the subtraction problem itself and the problem together with its solution.) Identify the atomic operation corresponding to this pair of states. In this case, let us just call it "subtract." Since these are the only states, we are finished.

F) Identify the "normal" domain of the operation(s) identified, using a variation of step C. Specifically, over what ranges of values can the initial problem state vary without affecting applicability of the atomic operation? Equivalently, what is (are) the corresponding atomic rule(s)? In the present case, the normal domain is just the domain of subtraction problems as specified above.

G) Identify the extra-domain condition, if any, which must be satisfied if this atomic rule (operation — plus domain and implied range) is to be applied. In the context of the intuitively known solution procedure, the "subtract" atomic rule applies to every element in its domain (i.e., to all pairs of numbers where the top one is not smaller than the bottom one). Hence, there are no extra-domain conditions.

Since there is only one atomic rule in the solution procedure, we are finished. The "subtract" atomic rule is THE required solution rule at this level. (Note: This subtract rule consists of the subtraction operation, together with the problem domain and goal.)

II. Reapplication of step-by-step analysis gives a more finely structured rule at the next level of top-down analysis. At this point, we may select any problem (including the original one) from the subtraction domain, which we have formally characterized in terms of the "--", "-----", and two number variables. (The unspecified solution variable constitutes the range.)

The first phase in refined analysis is to repeat the above steps (A—G) at a next level of detail.

A) What additional characteristics of the problems are referred to in applying the intuitively known solution procedure? In the case of column subtraction problems, columns clearly constitute additional characteristics because solutions proceed column by column.

B/C) Representation of the domain of subtraction problems thereby is enriched by allowing for a variable number of columns (from one up).

D) Next, the given problem is solved taking columns into account. That is, the subtract atomic rule (i.e., component) is refined. It is replaced with a sequence of states of the form

\[
\begin{array}{c}
xN \ldots x1 \\
- yN \ldots y1 \\
\end{array}
\]

\[
\frac{z1}{(zi-1) \ldots z1}
\]
each state having one more digit \( \Xi \) than the preceding state (indicating that one more column has been subtracted).

IMPORTANT: The \((x_i, y_i)\) column and the \((z_i-1)\) are encircled to distinguish these ACTIVE components of the problem (at state \(i\)) from those that are inactive. Active components are those that are attended to (i.e., operated on by the associated atomic rules) at any given step (state) of a problem solution. The "\(\Xi\)") denotes the goal to be achieved at state \(i\). (Notice at the initial level of analysis that both of the two given numbers are "active" in this sense.)

E–F) Next, the transition between the first pair of states is specified by the atomic rule "Go to the first column on the right and subtract." In turn, each successive state transition is specified by the atomic rule "Go to the next column to the left and subtract." The domain of this atomic rule is a problem schema. The active components in the schema are the just determined (last) digit in the partial difference (i.e., \(\Xi\)), which determines what the next column to the left is, and that column, which determines the column difference. As before, the result is a sequence of atomic rules.

G) Here, we see that the first atomic rule applies to every subtraction problem. There are no "extra-domain" conditions. Each of the succeeding atomic rules, however, is applied only when there is another column to the left (otherwise the problem is finished). Consequently, we introduce the extra-domain condition "There is a next column."

Suppose now that we have applied Steps D–G to just one problem (it could be the atomic one). At this point, we have a sequence of atomic rules, together with extra-domain conditions immediately preceding all but the first.

Where such extra-domain conditions exist, further analysis is called for.

H) Examine each of the extra-domain conditions to determine whether some succession of the following atomic rules ever generates an output state whose active components satisfy the extra-domain condition (as well as the domain of the first atomic rule following the extra-domain condition). The given problem has three columns, so there is another column after subtracting each of the first two columns.

Indeed, by choosing a suitable problem, this extra-domain condition can be satisfied any number of times. Irrespective of how many columns are chosen, there is always some problem with one more column. What this means, effectively, is that any number of applications of the atomic rule "Go to (i.e., attend to or activate) the next column and subtract the column" is possible (with some problem). Alternatively, reapplication is indicated until an output state is generated (e.g., a fully solved problem) that no longer satisfies the extra-domain condition — namely, "There is no next column." Moreover, when this extra-domain condition is no longer satisfied, the process STOPS so the underlying procedure is completely defined.

Whenever such reapplication (of some succession of atomic rules) is indicated, the extra-domain condition in question defines an iteration. Equivalently, the succession of atomic rules may be replaced by a loop, given a more efficient representation of the solution procedure.

More generally, one checks first to see if an extra-domain condition defines an iteration (i.e., loop). If it does not, and only if it does not, then it necessarily defines a selection.

III. Again, one can reapply step-by-step structural analysis to components of the Level II solution rule. As in the previous more informal top-down analysis, the "Subtract the column" atomic rule is one obvious candidate.

A) In this case, close attention is paid to what is done in subtracting individual columns, again building on the analyst's informal awareness of the solution procedure. In subtracting the first column of the given problem

\[
\begin{array}{c}
437 \\
-275 \\
\hline
162
\end{array}
\]

we get the partial answer (2) directly. In the next column, however, we must "borrow" in order to subtract (3−7) — where borrowing involves reference to the top digit (4) in the next column.

In effect, at this third level of analysis it is necessary to distinguish individual digits in the subtraction problem. (The digits then constitute the "additional" characteristics.)

B/C) Each of these digits, in turn, may be replaced with a variable ranging from 0 to 9 (subject to the overall constraint that the top numeral represent a larger number than the bottom one).

D) Next, we detail more fully the sequences of states associated with designated components (e.g., the subtract column atomic rule) of the previously determined solution procedure. This is accomplished by solving the above, more finely structured problem, giving explicit attention to the "additional" characteristics specified. Consider a case where the subtract column component rule is applied to a column which requires "borrowing." For example, consider the sequence of states

\[
\begin{array}{c}
4 \ 3 \ 7 \\
- \ 2 \ \_ \ 5 \\
\hline
\ 2 \ \_ \ 2
\end{array}
\]

\[
\begin{array}{c}
3 \ 4 \ \_ \ 7 \\
- \ 2 \ \_ \ 5 \\
\hline
\ 2 \ \_ \ 5
\end{array}
\]

\[
\begin{array}{c}
3 \ 4 \ 1 \ 3 \ 7 \\
- \ 2 \ 7 \ 5 \\
\hline
\ 2 \ 6 \ 2
\end{array}
\]

where the encircled entities are active and "\(\Xi\)" designates the goal at that state.

E/F) In turn, the successive pairs of states correspond to the atomic rules: "Go to the top digit in the next column and borrow," "Add the borrowed 10 to
problems — hopefully ones in which the previously unused conditions are satisfied. The result of each new analysis, then, must be combined level by level and component by component with the initial one. The process must be continued with new problems until the underlying procedure is completely defined. In this regard, notice that there always is a finite number of problems that will suffice for this purpose because the number of conditions in any procedure must be finite.

IV. Finally, let me emphasize that our analysis remains incomplete as regards borrowing across O’s. But this case too simply involves a selection (within the borrow selection). Since nothing new is required it is left as an exercise for the reader.

M. AUTOMATED STRUCTURAL ANALYSIS: TOWARD A MAN-MACHINE SYSTEM

With this example in mind, let us consider how the process of top-down step-by-step analysis might be partially automated. The intent is that a person knowledgeable in the subject matter and familiar with the capabilities of the target student population would be able, with computer assistance, to carry out the process reliably with no special or independent knowledge of the cognitive instructional theory, of which the process is a part.

To make this possible, major portions of structural analysis should be carried out automatically, relieving the subject matter-pedagogue specialist (i.e., the analyst) of theoretical concerns. The analyst should be required to provide only information relevant to the content and learner capabilities. Use of that information in constructing the desired solution rule should be performed independently by a theoretical expert, preferable an automated expert—ideally, say, one realized electronically as a computer program.

Specifications for such a procedure are represented graphically in Figure 3. In this “man-machine” system, at various stages of analysis, the machine (e.g., computer) requires specific information from the human analyst and operates on that information in accordance with structural analysis principles. The important thing about this procedure is that the human analyst need not be familiar with structural analysis per se. It is sufficient that the analyst be conversant with the subject matter and the minimal capabilities of the targeted student population.
N. EXTENSION TO COMPLEX PROBLEM DOMAINS

As mentioned in Part One, structural analysis traditionally has been used in conjunction with arbitrarily complex problem domains. To be sure, the analysis of simple domains of the sort we have been considering (i.e., a priori problem domains which are equivalent to domains associated with sampled problems) is involved in analyzing all domains. Where the a priori domain includes more than one simple problem domain, however, further analysis is required.

The simplest extension of structural analysis comes in where the a priori domain is a simple set union of a finite number of different, but non-overlapping problem domains, which collectively are equivalent to the a priori domain. The domain of grammatical rules taken collectively, or even the basic computational problems in arithmetic, may be viewed in this way — as a simple union of discrete domains of problems. As we have seen in the case of subtraction, for example, there is a single solution rule underlying each of these domains.

Unlike most grammatical rules, however, which tend to be quite distinct, computational rules in arithmetic tend to be hierarchical. Thus, for example, the multiplication algorithm (rule) assumes the addition rule as an (atomic) component. Similarly, division assumes multiplication and subtraction. In effect, the former are more complex rules which can be generated via higher-order rules operating on the latter together with other (lower-order) rules (e.g., see Scandura, 1977).

Most of the structural analyses described in Part One involve a priori domains which are still more complex — both with regard to the nature of relationships among individual rules and the explicit and general nature of the higher-order rules involved. It is not possible, for example, to characterize the domain of geometry construction problems (Scandura, Durin, & Wulfeck, 1974) or that of generating algebraic proofs (Scandura & Durin, 1977 in terms of any finite number of problem types (i.e., simple domains), no matter how large the number. One will always be able to find some problems that naturally fall into the category of geometry construction problems or of algebraic proofs, and hence in the a priori domain, but which do not belong to one of the previously defined problem types.

To deal with a priori domains of this type, the analyst begins by selecting a representative sample of problems from the a priori domain. Each of these is then analyzed by the above methods, yielding a finite set of solution rules for solving each of these problem types.

The key to further analysis is the observation that each of the obtained solution rules corresponds to a higher-order problem. Specifically, each rule may be viewed as the solution to a higher-order problem. For example, a rule for solving a geometry construction problem may be viewed as the solution to the higher-order problem of constructing a solution rule for a specific type of geometry.
construction problem. Similarly, the subtraction rule we have been discussing may be viewed as the solution to the higher-order problem of devising constructing a solution algorithm for subtraction problems.

Let us consider, for example, the broad domain of tasks which might reasonably be termed computational. Clearly, this a priori domain is indefinitely large in scope and could reasonably include everything from subtraction problems to complex engineering techniques, and need not even be restricted to mathematical or scientific tasks. Suppose further that we have selected a representative sample of tasks from this domain and have analyzed each as above, giving a finite set of solution rules. Each of these solution rules, in turn, would next be reformulated as higher-order problem. The subtraction rule, for example, would be converted in the analyst's mind into the higher-order problem of constructing a solution rule for subtraction problems.

Representing such a problem formally, of course, requires specifying the needed information (the given) as well as the goal. Specifically, the analyst must specify those characteristics of such problems, in this case of higher-order problems, together with known rules, which collectively make it possible to achieve the goal (i.e., the previously determined solution rule). To accomplish this, the analyst must be thoroughly familiar with the content.

In the case of subtraction, for example, the meaning of subtraction as taking away and the place value concept of representing numbers are crucial. More particularly, before teaching an arithmetic "algorithm" itself, mathematics educators generally recommend teaching the equivalent of a rule for performing the corresponding operation (e.g., subtraction) with concrete objects represented in a standard place value format. Thus, for example, given a pair of numbers, say 132 and 27, represented concretely as

```
| 30 | 20 | 7 |
```

The student is shown how to take away the amount represented by the smaller quantity from the larger by taking away from the larger quantity as many groups of each size as there are in the smaller quantity. Where the number of groups of a particular size (e.g., ones) in the larger quantity is smaller than that in the smaller quantity, the student learns to take away by first converting a larger grouping (e.g., tens) in the larger quantity into a smaller grouping (e.g., one ten's group would be converted into ones in the larger quantity so that the 7 ones in

the smaller quantity might be taken away). Implicitly, the student also learns to begin work with the smaller place values (groupings) and to work toward larger ones.

Normally, students are not taught general rules (procedures) for performing arithmetic operations on concrete objects in a systematic way. Rather, students gradually acquire an informal awareness of such rules by solving a variety of specific concrete problems, with concrete objects and/or pictorial representations of such objects. Dienes' blocks (e.g., Dienes, 1973) are commonly used for this purpose.

To summarize, the higher-order problem in the case of subtraction would consist of a concrete realization of the subtraction algorithm as the problem given, and a syntactic solution rule as the goal. (The solution to this higher-order problem, of course, is the subtraction algorithm, i.e., rule.) Although students normally are not taught how to make the transition from concrete to symbolic, our experience suggests that it would be a good idea. Many students who can perform both the concrete manipulations and the syntactic algorithm itself fail to understand the relationship between them. Not knowing which concrete manipulations correspond to which steps in the column subtraction algorithm, results in the two skills being cognitively independent. Consequently, these skills do not reinforce one another in the sense, for example, that one might be derived from the other. This type of relationship can be very helpful, for instance, in cases of partial recall. (Knowledge of relationships corresponds to higher-order rules. Teaching such relationship has been explicitly incorporated in a computerized arithmetic tutorial system we have developed.)

In general, higher-order problems will be less concrete than lower-order ones, and correspondingly, specifying them will require deeper insight into the subject matter. This method of formally representing the domains of higher-order problems and of identifying underlying higher-order solution rules, however, is the same as that used with simple problems. Thus, structural analysis takes higher-order problems as a new starting point and proceeds from there.

First, the problem domain, corresponding to each high-order problem, is determined as before by substituting variables for the given. There are two basic directions over which the above higher-order problem might naturally be generalized: 1) the numerical base used in the place value system (base 10, 2, 8, etc.) and 2) the binary operation to be performed (e.g., take away, combine, combine with itself a given number of times (x), determine how many times a quantity can be taken away from a standard (f)).

For present purposes, we ignore the former and restrict ourselves to a simple higher-order rule that converts concrete manipulations for combining and taking away, respectively, into syntactic algorithms (i.e., rules) for column addition and subtraction. (The process can easily be generalized to different operations and bases.) Without bothering to detail each step in the structural analysis, such
Analysis yields correspondences (i.e., atomic rules) between combining quantities and the basic addition facts, taking away and basic subtraction facts, converting from larger place values to next smaller ones and “borrowing”, and converting from smaller place values to next larger ones and carrying.

These correspondences are all that a mathematics teacher would be likely to teach to students directly. There are several reasons why more detailed instruction concerning relationships is not common. First, the concrete manipulations children learn are normally never brought to the same degree of formal awareness as is the case with the more familiar syntactic algorithms. Hence, students are rarely taught how to actually construct (i.e., derive) syntactic arithmetic solution rules, and even less often how to formally represent (describe) them. To do so, would among other things require learning a process for externalizing, or describing known rules. Such skills are rarely taught explicitly.

In the analytical research described in Part One, structural analysis was a largely intuitive process. It is, therefore, instructive to see how the more refined formulation above corresponds to the process as originally described (see Section A, Part One). In applying structural analysis to arbitrary domains, the first step is essentially the same. The analyst selects and/or constructs a representative sample of problems from the a priori domain. This is something the analyst does intuitively and the above systematization adds relatively little. In the earlier work, however, we glossed over the process of constructing solution rules corresponding to each of the representational problems. As we have seen above, this fundamental process is relatively complicated when carried out objectively and systematically – in a manner which is generally applicable.

The third step of identifying parallels among solution rules and representing them as higher-order rules can now be seen to involve two things. The parallels referred to earlier correspond essentially to specifying the domains of higher-order problems (which correspond to identified solution rules). The basic question is which of the identified solution rules might reasonably be associated with the same higher-order problem domain, and hence, be viewed as “parallel.”

Equivalently, solution rules are parallel just because they can be generated via the same higher-order rule (applied to different higher-order problems in the same domain). Actual specification of the domains and procedures of the corresponding higher-order rules, of course, proceeds as described previously.

The fourth step of structural analysis, involving the elimination of redundant rules (i.e., rules which can be derived from other rules), has been a largely mechanical procedure from the beginning so there has been little to add. However, the last step described in Part One, of testing and refining the set of rules obtained at any given level of analysis (i.e., before moving to the next higher level), has been made largely obsolete by the present method of structural analysis. Having tentatively completed a structural analysis using the above method, (relative) completeness requires that the analyst determine whether there are problems in the intuitively specified a priori domain which are not accounted for by the existing rules but which nonetheless are believed to be important by the analyst.

References


