Donchin, E., Ritter, W., & McCallum, W. C. Cognitive psychophysiology: The endogenous components of the ERP. In E. Calloway, P. Tueting, & S. Koslow (Eds.), *Brain event-related potentials in man*. New York: Academic Press, 1978.

Estes, W. K. Component and pattern models with Markovian interpretations. In R. R. Bush & W. K. Estes (Eds.), *Studies in mathematical learning theory*. Stanford, Calif.: Stanford University Press, 1959.

Falmagne, J. C. Stochastic models for choice reaction time with applications to experimental results. *Journal of Mathematical Psychology*, 1965, *2*, 77-124.

Falmagne, J. C., & Theios, J. On attention and memory in reaction time experiments. *Acta Psychologica*, 1969, *30*, 316-323.

Gagné, R. M. The acquisition of knowledge. *Psychological Review*, 1962, *69*, 355-365

Gagné, R. M. *The conditions of learning* (2nd ed.). New York: Holt, Rinehart & Winston, 1970.

Greeno, J. G., James, C. T., DaPolito, F., & Polson, P. G. *Associative learning: A cognitive analysis*. Englewood Cliffs, N.J.: Prentice-Hall, 1978.

Halff, H. M. The role of opportunities for recall in learning to retrieve. *American Journal of Psychology*, 1977, *90*, 383-406.

Kohler, W. *Gestalt psychology: An introduction to new concepts in modern psychology*. New York: Liveright, 1947.

Lewis, G. W., Rimland, B., & Callaway, E. *Psycho-biological predictors of success in a Navy remedial reading program* (TR 77-13). San Diego, Calif.: Navy Personnel R & D Center, 1976. (NTIS No. AD A037 339)

Link, S. W. The relative judgment theory of two choice reaction times. *Journal of Mathematical Psychology*, 1975, *12*, 114-135.

Link, S. W., & Heath, R. A. A sequential theory of psychological discrimination. *Psychometrika*, 1975, *40*, 77-105.

Minsky, M. A framework for representing knowledge. In P. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill, 1975.

Norman, D. A., Gentner, D. R., & Stevens, A. L. Comments on learning schemata and memory representation. In D. Klahr (Ed.), *Cognition and instruction*. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1976.

Ollman, R. Fast guesses in choice reaction time. *Psychonomic Science*, 1966, *6*, 155-156.

Schank, R., & Abelson, R. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1977.

Shoenfeld, A. H. *Problem solving strategies in college level mathematics*. Berkeley, Calif.: The Group in Science and Mathematics Education, University of California, 1978.

Weigel, G. A., Schendel, J. D., & Halff, H. M. *Cued recall for words presented in separate pairs*. Paper presented at the 18th annual meeting of the Psychonomic Society, November 1978.

Wescourt, K. T., Beard, M., Gould, L., & Barr, A. *Knowledge-based CAI: CINS for individualized curriculum sequencing* (Tech. Rep. No. 290). Stanford, Calif.: Stanford University, Institute for Mathematical Studies in the Social Sciences, 1977.

Yellott, J. Correction for guessing in choice reaction time. *Psychonomic Science*, 1967, *8*, 321-322.

Yellott, J. Correction for fast guessing and the speed-accuracy tradeoff in choice reaction time. *Journal of Mathematical Psychology*, 1971, *8*, 159-199.

In R.E. Snow, P.A. Frederico & W.E. Montague (Eds.) Aptitude Learning and Instruction: Cognitive Process Analyses. Hillsdale, NJ: Erlbaum. 1980

18 Planning Nets: A Representation for Formalizing Analogies and Semantic Models of Procedural Skills

Kurt VanLehn
John Seely Brown
Xerox Palo Alto Science Center

INTRODUCTION

At some time in our lives, we have all been forced to learn the procedural skills that supposedly comprise mathematical literacy (e.g., place-value addition) through the process of rote memorization, perhaps, enhanced by the use of "models" (e.g., the abacus). These models were intended to provide an intuitive basis for a given procedure. But what really is a "model" of a procedural skill? How does it help in learning? How faithful can it be made to be? And, more generally, how can it help a procedure take on "meaning"?

This chapter is directed at understanding how procedures can take on "meaning." It is intended to provide a small step in that direction by discussing a particular kind of "semantics" for procedural skills, which we call *teleologic* semantics, in the context of the unambiguous and totally specifiable procedural skills of elementary mathematics.

The teleologic semantics of a procedure is knowledge about the purposes of each of its parts and how they fit together. Such knowledge is the province of true masters of the procedure. Its value is extolled by the proverb, "To really understand something, one must build it." Teleologic semantics is the meaning possessed by one who knows not only the surface structure of a procedure but also the details of its design.

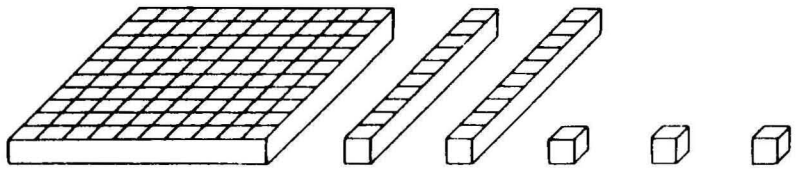
This chapter has two arguments. First, we motivate the particular representation that we use for teleologic semantics, which we call *planning nets*, by showing how it can capture analogies between procedures as seen by an expert at those procedures. Second, we show that teleologic semantics, as formalized by planning nets, is useful by describing several potential applications in the field of

education. In particular, some consideration is given to how teleologic semantics can be explained and to how it provides a useful framework for developing "optimal" sequences of "model" procedures (or microworlds) for guided discovery learning.

Analogy Between Procedures

Before we delve into a technical discussion of procedural analogies, let us consider a simple example of an analogy between the procedure for adding two multidigit numbers and a "model" procedure for addition that manipulates physical objects that represent numbers. The model procedure is a physical procedure in that it manipulates physical objects that stand for numbers. Before we can describe the procedure, we briefly describe the objects that it manipulates, namely, Dienes Blocks.

The Dienes Blocks Representation of Numbers. Dienes Blocks provide an explicit representation of base-10 numbers—namely, a set of *unit* blocks for representing the units; a set of *long* blocks consisting of 10 unit blocks molded into a long stick for representing the 10s; a set of *flat* blocks consisting of 10 long blocks laid next to each other, thus forming a 10 × 10 square for representing the 100s; and finally a set of *cubes* in the form of 10 × 10 × 10 units for representing the 1000s. A number (of four or less digits) can be physically represented by selecting the number of unit blocks to correspond to the units digit, the number of long blocks to correspond to the 10s digit, and so on. Hence a particular multidigit number is represented by piles of units, longs, flats, and cubes. Here, for example, is 123 represented in Dienes Blocks:



The base-10 nature of the symbolic place-value scheme for representing numbers is then made explicit, since one can see the direct translation of a number represented as piles of Dienes Blocks into a base-1 system (i.e., the total number of units comprising all the blocks in all the piles).

Dienes Block Addition. Addition of two multidigit numbers represented as concrete Dienes Blocks involves forming set unions and "trading." The units pile for each of the two numbers is first unioned together. This corresponds to adding the units column. Next, the resulting set is examined. If it contains more than 10 unit blocks, then 10 blocks are removed from this set and traded for a long block (consisting of 10 units), which is then placed in a pile of long blocks of the top number. This corresponds to carrying from the units to the 10s column in standard addition. The procedure now repeats, unioning the longs piles, then the flats, and so forth.

A theory of analogy between procedures, applied to this case, should be able to capture not only the fact that Dienes Block addition and standard addition produce the same answers given the same inputs, but that their *internal structures* correspond as well. Set unions match with column sums, trading matches carrying, and so on.

Two-Pass Addition Illustrates Differences in Closeness. We were recently struck by the way Dienes Blocks were being used in a school. In particular, the Dienes Blocks procedure being taught was not as described earlier but instead had the students combining all the piles of blocks together and then returning to the units pile and trading up and so on. Thus, in standard multidigit addition, a carry is (potentially) performed after each column operation, whereas in this version of Dienes Block addition, the "trading" (or carrying) operation was being deferred until all the columns has been initially processed. One intuitively feels that this second, two-pass procedure is not as closely analogous to standard addition as the previous, one-pass Dienes Block procedure.

A theory of analogy should have some formal measure that can predict how close an analogy is. The theory that follows has such a formal mechanism, called a *closeness metric*. The degree of correlation between the predictions of the closeness metric and subject's intuitive judgments of closeness is one verification condition for the theory.[1]

Why Arithmetic? The examples in the chapter are all drawn from the computational procedures of arithmetic, even though the techniques we have developed have wide applicability. We limited our examples to arithmetic for several reasons. Everyone knows how to add and subtract, so lack of familiarity with the example domain will not hinder comprehension of these admittedly rather abstract formalisms. Arithmetic is a highly evolved, complex system of procedures. It has iteration, recursion, tables of facts, and, of course, a rather nontrivial data representation—namely place-value numbers. Lastly, arithmetic is taught in school. This means our theories are more likely to accrue the benefits of thoughtful, experience-based criticism from those with a sincere interest in putting the theories to work.

[1] It is safe to assume that individuals will differ in their judgments of the closeness of analogies. We take the position that this is due to the different deep structures that they assign to procedures. For example, someone who is just learning addition may not find the analogy between one-pass and two-pass addition particularly close. This might be due to a lack of distinct concepts for "carrying" and "column addition." So how one understands a procedure affects the data against which the theory of analogy will be verified. Because we are interested in teleologic semantics and because teleologic understanding is a mark of expertise, it is important to use experts as subjects.

Organizational Overview. The chapter is divided into three parts. The first part is an exposition of some of the basic concepts of formal theories of analogy. We assume that an analogy can be represented as a *mapping* between a *deep structure* representation of each procedure that is expressed as a *maximal partial isomorphism* between the two deep structures. Thus, after an analogy has been comprehended, we would expect to find cognitive structures that could be modeled by three components; two of which represent the abstraction or deep structure of the two procedures, and the third representing the structure-preserving map (i.e., analogy) between these two structures.

The second part of the chapter motivates the planning-net representation of teleologic semantics by using it as the *deep structure* component of a theory of analogies between procedures. The third part is an examination of some of the applications of this theory to education. In particular, we discuss a paradigm for explaining the teleologic semantics that involves using a sequence of analogies such that each analogy illustrates exactly one concept underlying the synthesis of the given "target" procedure (e.g., place-value subtraction). This paradigm is then augmented with a set of "naturalness" principles for structuring a sequence of analogies, thereby addressing the problem of how to design an optimal sequence of "microworlds" or models for enhancing discovery learning.

We caution the reader that our style of arguing with examples has led to the incorporation of a great deal of detail into the subsequent pages. However, if artificial intelligence has contributed anything to cognitive psychology, it is an appreciation that ignoring trivial detail often leads to overlooking nontrivial problems.

A GENERAL THEORY OF ANALOGY

This section presents a theory of analogy so general that it is almost vacuous. It appears that virtually any theory of analogy, including the theory of procedural analogies that is presented later, can be recast as a special case of this general theory. Thus, this general theory is apparently immune to refutation. Nonetheless, it allows discussion of some concepts common to all analogies, such as "closeness," before becoming immersed in the details of procedures and their representations.

Mapping Between "Deep Structures"

We view an analogy as a comparison of two "things" that can be broken down into three parts: (1) an analysis of the first thing into some abstract description (or deep structure); (2) an analysis of the second thing into another abstract description, and (3) a mapping between the two descriptions. This tripartite breakdown is the foundation of the general theory of analogy. Exactly this breakdown is also found in Tversky's work on similarity, a domain that illustrates the general theory more clearly because of the simpler "deep structures" that are used (Tversky, 1977).

Much research on similarity has used pairs of geometric figures or letters. A typical task is to rate the similarity of o to c. Tversky's analysis of this task is to assume a feature space, describe each figure as a set of features, then predict the similarity judgments with some "metric" on the overlap of the feature set of o with the feature set of c. The correlation of the judgments with the predictions serves as a verification condition on the feature space and the metric. Often, the features are not very abstract; o might be mapped into the description {curved, circular, closed}, and c would become {curved, circular, open}.

Much of the research on analogy has studied a task one often finds on intelligence tests—namely, to fill in X in a statement of the form: "A is to B as C is to X." Most commonly, the four elements A, B, C, and X are either words or geometric figures. A simple example of a word analogy problem is: "Red is to Stop as Green is to (a. Go; b. Halt)." Superficially, this appears to be a different sort of task than the similarity task, since there are four things rather than two. But the two tasks become very much the same when one considers the analogy task to be a comparison of *relationships* rather than directly apprehendable things. This is a widely held view of analogy. Indeed, the instructions to one analogy test, as quoted by Evans (1968), read: "Find the rule by which Figure A has been changed to make Figure B. Apply the rule to Figure C. Select the resulting figure from Figures 1 to 5 [p. 272]."

Actually, these instructions represent just one strategy for answering analogy problems. Evans' ANALOGY program, for example, used a different strategy, whereby it extracted an AB rule, then found five rules for pairs C1, C2, C3, C4, and C5, then finally chose one rule of the five as being the most similar to the AB rule. The existence of many different strategies for solving analogy problems also obscures the parallels of this task to the similarity and metaphor tasks. Yet when one is done finding the analogy, one possesses the same three maps; an abstraction from AB, an abstraction from CX where X is the chosen answer, and the partial match (or mapping) between the two resulting abstract descriptions.

In short, if one ignores the strategic differences between solving an analogy and evaluating a similarity, and if one puts relationships on an equal footing with letters and geometric figures, then there is very little difference between the analogy task and the similarity task. After either task is completed, the cognitive structures can be modeled by three components: the two abstract descriptions and the mapping (in the form of a match) between them.

Basic Definitions

In this subsection, several basic concepts are discussed. They all follow rather immediately from the three-task view of analogy already described. As earlier,

they are motivated and illustrated with examples from Tversky's theory of similarity.

Intersection and Difference Sets. A good way to summarize the outcome of the matching map is in terms of one *intersection set* and two *difference sets*. As an example, take the similarity task mentioned earlier to evaluate the similarity of o and c. Their descriptions, let's say, are the feature sets {round, curved, closed} and {round, curved, open} respectively. Call these sets A and B, the abstract descriptions of o and c. Then, the intersection and difference sets are:

$$A \cap B = \{\text{round, curved}\}$$

$$A - B = \{\text{closed}\}$$

$$B - A = \{\text{open}\}$$

This is not particularly startling, to be sure. Indeed, there are stereotypical ways of referring to these sets in English similes: "A is like B in that $A \cap B$," or "A is like B except that $A - B$ instead of $B - A$."

Maximal Partial Graph Morphisms Generalize the Notion of "Match." With more complex languages than feature spaces for expressing abstract descriptions, one must of course give a new definition of "match." For example, consider the analogy (from Sternberg, 1977): "Washington is to 1 as Lincoln is to 5." Suppose semantic nets are the representation language. The abstract description of the relationship Washington:1 is a certain chain of semantic links from the node "Washington" to the node "1." The description of Lindoln:5 is a different chain. However, when one finally finds the correct way to view the two relationships (which is rather nontrivial for this example), then the two chains end up bearing the same sequence of link names—namely: Last-name, image-of, portrait-on, dollar-amount. That is, "Washington" is the *last name* of the man NODE_{31}; the *image of* NODE_{31} appears in the picture NODE_{7}; NODE_{7} is the *portrait on* the kind of dollar bill NODE_{68}; and the *dollar amount* of NODE_{68} is "1." The chain for the Lincoln:5 relationship is a completely distinct chain, but it has exactly the same sequence of link labels. In this sense, the analogy is perfect.

To make these two chains match, the definition would have to be sensitive to: (1) the order of the links; and (2) the labels on the links. A definition in terms of intersection of sets of links would be inappropriate because none of the links are identical and because such a definition would ignore the topology of the descriptions. A definition of "match" that is appropriate for semantic nets (or any other representation with the topology of a labeled directed graph, including planning nets) can be defined in terms of a graph isomorphism:

Adjacency: Two links of a graph are adjacent if they are incident with a common node.

Isomorphism: An isomorphism of labeled directed graphs is a one-to-one correspondence on the links that preserves the adjacency, direction, and label of the links.

The "match" of the two semantic-net chains X and Y can now be defined to be the *maximal* graph isomorphism from a subgraph (subsequence) of X to the subgraph of Y. By "maximal," we mean the isomorphism that pairs the largest number of links correctly. Unfortunately, use of maximality precludes any mathematical guarantee of the uniqueness of the resulting isomorphism. However, in practice, we have yet to be plagued by a nonunique maximal isomorphism.

Note that we have defined "match" as a map that is an isomorphism between *sub*graphs of the two deep structures. The map between deep structures is not necessarily total (i.e., onto) in either direction (we are in the process of investigating a revision of this aspect of the definition as well as the interesting situation where it is many-to-one and hence would have the properties of a homomorphism). In other words, the analogy is a mapping that is a *maximal partial graph isomorphism*. However, we abbreviate our terminology somewhat and say that the analogy from A to B is formalized by the mp-morphism from A to B (i.e., we speak of the analogy as being this structure-preserving map).

To replace the terms *intersection set* and *difference set*, we simply use *intersection subgraph* and *difference subgraph*. There are, of course, two difference subgraphs for an mp-morphism—namely the residue portions of each of the deep structures being compared. Throughout this chapter, we continue to use the symbology of sets for these concepts, even though the designated entities are not sets, but subgraphs.

Closeness Metrics. Both the similarity task and the analogy task involve the ranking of the match between two things or, rather, between their abstract descriptions. The subject is asked to rank the degree of similarity or choose the closest analogy. We assert that both kinds of judgments can be modeled by a function over the intersection set (or subgraph) and two difference sets (or subgraphs). In similarity research, this three-argument function is often called a "similarity metric," even though there are cases when the function is not a proper mathematical metric (see Tversky, 1977). With the same sloppiness, we call the function that ranks the closeness of analogies a *closeness metric*.

These metrics can be rather complex. Certain features might be more salient than others, and one might model this difference by giving the former more weight in a summation over the various sets. These metrics might even be asymmetric,[2] which means they are not proper "metrics" in the strict mathemat-

[2]Tversky (1977) weighted the features in the set $A-B$ more heavily than the features in the set $B-A$ in order to account for certain experimental data – for example, that "Red China is similar to North Korea" has a lower degree of intuitive similarity than "North Korea is similar to Red China."

ical sense. In short, determining the intersection set and the two difference sets is not the end of the story for predicting similarity judgments; the metric can play a decisive role.

Monotonicity, etc. We take the position that a precise statement of the closeness metric for procedural analogies can only be determined from detailed empirical studies. However, Tversky has shown that if certain formal conditions on the metric can be guaranteed, such as its monotonicity over subsumption of the intersection and difference sets, then the metric can have a simple, linear form (Tversky, 1977). (One of us—VanLehn—has investigated some of the conditions for procedural analogies and will discuss them in a later report.)

Individual Differences and Learning. We have been speaking of *the* abstract description (or deep structure) of a thing as if this object were the same for all people. In some tasks, such as assessing the similarity of letters, it seems reasonable for literate individuals to have roughly the same representation language and the same abstraction functions for extracting descriptions from the letters. But this assumption is rather implausible in many other cases. In these cases, individual differences in conceptions of the things being compared is likely to influence judgments of the closeness of analogy. This would make verification of a theory significantly more difficult.

Individual differences affect analogy, but analogy also affects the individual differences. That is, one can learn from analogies. More specifically, when an individual understands an analogy, he or she may become aware of descriptive features that were previously overlooked. So a complete theory of analogy must allow for an evolution of an individual's conception of the things being compared over the course of testing.

In this research, we ignore these difficult methodological problems by assuming that the subjects who are judging the closeness of the analogies are *experts*. That is, they all have a complete representation of the things being compared and, hence, can be assumed to have roughly the same representations. Secondly, they already know all there is to know about the things being compared and therefore learn very little over the course of the testing.

FINDING THE RIGHT REPRESENTATION FOR PROCEDURAL ANALOGIES

In this section, several candidate representations for procedures are examined as a basis for a theory that predicts the closeness of procedural analogies.[3] Possible

[3]The judgments of closeness are those of experts on arithmetic and so can be taken to reflect the [illegible] semantics of arithmetic.

representations range from a very superficial one—namely, a simple chronological list of actions—on up to a very abstract representation that involves goals, constraints, and other planning knowledge—namely, planning nets. Our research has shown that planning nets are the only serious contender, so the discussion of the others is quite brief. However, the more superficial representations are mentioned in this section for a reason—namely, to show how a human (or machine) can construct a very abstract representation of a procedure by ascending through several levels of representation. We do not claim that the structure of this section models the abstraction process that a person executes when assimilating a procedural analogy, but it does provide an indication of the complexity that such a process would have to have.

Traces

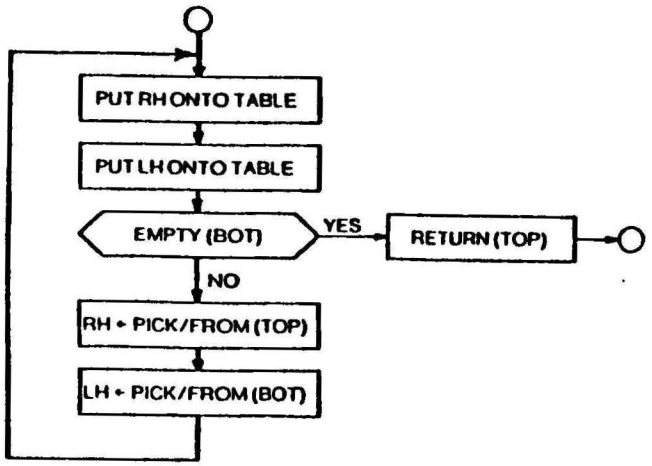
The *trace* of a procedure is simply a chronological list of the actions it performed during one particular execution. This representation of a procedure can be constructed directly from observation of the execution of the procedure (although there are the usual problems in choosing the "grain size" of primitives).[4] However, traces are a highly inappropriate representation for procedures, as the following example indicates.

Consider an analogy between Dienes Block addition and written addition. These two traces would probably have few, if any, action labels that match. The action "write '4'" would have to be matched against a group of four actions labeled "place one block in the pile," whereas the action "write '8'" would have to be matched against a group of eight block-placing actions. Such sophisticated matching could not be represented by an mp-morphism. Indeed, the match seems to require the *concept* of "write *n*" and the *concept* of "repeat single block placement *n* times." These are abstractions over action sequences and so should be part of the representation rather than the matching mechanism. Incorporating such concepts into the representation lifts us to the next level of abstraction.

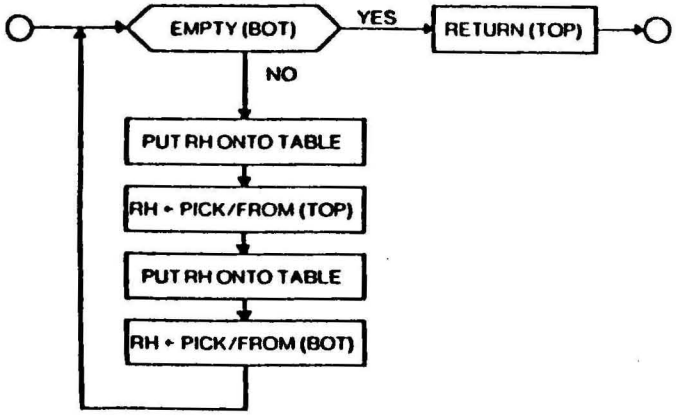
Flowcharts

By generalizing over a large collection of traces, one could derive a notion of the observed procedure that could be represented with a programming language,

[4]The folklore about protocol taking, supported by a few experiments (Card, 1978), is: When in doubt, use a finer grain size. If the grain size is too large, one might miss distinctions. If one errs the other way and makes the grain size too fine, then one creates more work for oneself; yet if one is tenacious, the relevant distinctions will ultimately appear, probably as relations between *groups* of actions instead of single, individual actions. So it appears that the grain-size issue (and a very similar issue—the choice of primitive actions) is more of a practical trade-off than an insurmountable source of uncertainty in the theory.



FLOW CHART FOR A BASE-1 BLOCK SUBTRACTION PROCEDURE USING TWO HANDS



FLOW CHART FOR A BASE-1 BLOCK SUBTRACTION PROCEDURE USING ONE'S LEFT HAND

such as flowcharts. Granted, this generalization would be nontrivial: Repetitious sequences of actions would become loops; objects that are manipulated similarly become the contents of variables, and so on. Nonetheless, constructing a program from examples is well within human ability.

However, flowcharts would also be a poor representation for analogy. Consider a simple subtraction procedure for numbers represented as base-1 blocks as illustrated by the lower flowchart on pg. 104. The primitive terms used in this flowchart are as follows: LH stands for someone's left hand. TOP and BOT stand for place mats on the TABLE. The BOT set of base-1 blocks is subtracted from the TOP set of blocks by pairing off a block from each, using the primitive actions PICK/FROM and PUT/ONTO and tossing them onto the table. When the bottom "number" is "zero" (i.e., empty), whatever is left in the top "number" is the answer. However, notice that by merely shuffling the order of the steps somewhat and using two hands instead of one, a new procedure can be constructed that is intuitively very similar to the old procedure; yet its flowchart (see pg. 104) shares virtually no isomorphic subgraph with the old procedure's flowchart. Because the intersection graph is so small relative to the difference subgraphs, a reasonable closeness metric would have to report that the two procedures are not very close—a false prediction. So for this and other reasons, flowcharts also seem to be a poor representation or level of abstraction for procedural analogies.

Procedural Nets

On the basis of the foregoing example, it might appear that flowcharts are too committed to a set order of performing steps, since the two base-1 flowcharts have the same steps but order them slightly differently. Also, these charts lack the typical hierarchy of subprocedures that is used in computer programs to modularize and organize the procedure. This suggests using a structure that emphasizes the subprocedure hierarchy and deemphasizes the temporal sequence of subprocedures.

Just such a structure has been developed for modeling children's bugs in arithmetic procedures—namely, BUGGY's *procedural-net* representation (Brown & Burton, 1978). Although we do not pause here to explain this representation, a procedural net for a very familiar procedure—namely, standard subtraction—is included as Fig. 18.1. However, procedural nets also fail as a basis for a theory of analogy, as illustrated in the following example.

Consider two Dienes Block subtraction procedures: (1) in "big-pile" Dienes Block subtraction, a number is represented by one big pile of Dienes Blocks; (2) in "sorted" Dienes Block subtraction, all the blocks are kept sorted into little piles according to their shape. Intuitively, these two procedures are quite closely analogous. But when the procedural nets are formed and the matching is done, we find the following statistics:

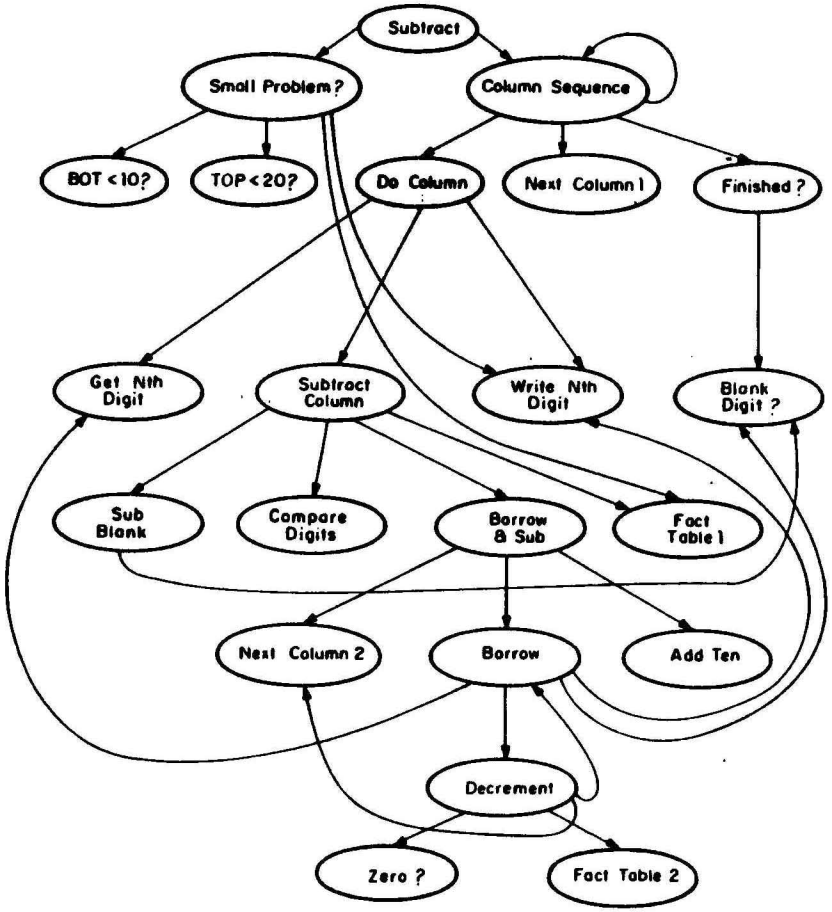


FIG. 18.1.

$A \cap B$ contains 6 nodes.

$A - B$ contains 10 nodes.

$B - A$ contains 16 nodes.

The intersection subgraph is far too small compared to the difference subgraph for this analogy to be rated "close" by any reasonable metric. So again, we must abandon a representation and look for a higher level of abstraction.

Planning Knowledge Seems Necessary

Both flowcharts and procedural nets are at the "program" level of abstraction. That is, they both are close to the sorts of languages one sees for computer programs. The problem with this level of abstraction seems to be that some design decisions that do not seem so consequential to the intuition have an enormously large effect on the "program." The framework that analogy seems to require is something that extracts these sorts of choices out of their final manifestation, makes them explicit, and relates them in a reasonable way to other, more important elements of the design. In short, what seems necessary is a representation of the design process behind a procedure—this allows one to say which choices are important and which are relatively minor. The process of creating a procedure from a set of constraints is traditionally called "planning" by the artificial intelligence community. So the abstract representation that analogy seems to require appears to involve planning knowledge and planning inferencing.

Planning knowledge includes not only the functional decomposition of the surface structure of the procedure but also the reasoning that was used to transform the goals and constraints that define the *intent* of the procedure into its actual surface structure. The formalism we use to represent this knowledge, we call *planning nets*. These planning nets are an extension of Sacerdoti's pioneering work on representing procedural knowledge for robotics (Sacerdoti, 1977). Before presenting the formalism (which lies at the heart of the remaining parts of the chapter), it is best to get some idea of what this "planning knowledge" is that is going to be incorporated into the representation. To this end, we plan out a very simple subtraction procedure, called "base-1 blocks subtraction," that represents a number as a pile of unit blocks. Later, we show how planning nets capture this knowledge in a summary form.

Constraints and Planning Heuristics

The basic idea of formal planning is to take a declarative, rulelike presentation of the goals of the procedure and the world in which it is to be implemented, and transform them into a surface structure that achieves the goals while remaining inside the constraints imposed by the world. There is always an element of common sense in planning, and as this is formal planning, use of common sense must also be recorded.

These two knowledge sources are called *constraints* and *heuristics*. Both can be represented as pattern-action rules in some suitable formal language, but for our purposes, English will suffice.

The constraints that characterize base-1 blocks subtraction are listed next:

1. Goal: If EMPTY (BOT) then return TOP as the answer (i.e., $n - 0 = n$).
2. The decrease in TOP must EQUAL the decrease in BOT (i.e., a recursive definition of subtraction).
3. a is EQUAL to b (i.e., all blocks are equal).
4. Over the action (Y ← PICK/FROM(X)), the decrease in X is EQUAL to the increase in Y (i.e., blocks are conserved over the picking-up action).
5. Over the action (PUT Y ONTO X), the increase in X is EQUAL to the decrease in Y (i.e., blocks are conserved over the putting-down action).
6. The action (Y ← PICK/FROM(X)) requires EMPTY (Y) beforehand (i.e., the hand must be empty before picking up a block).
7. The action (PUT Y ONTO X) entails EMPTY (Y) afterwards (i.e., the putting-down action always empties the hand completely).
8. ~ EMPTY (X) before the action (Y ← PICK/FROM(X)) entails that afterward, there exists a such that a is the contents of Y (i.e., the hand picks up exactly one block).

The meaning of the primitives is as follows. TOP and BOT are place mats on the TABLE. The subtraction problem $n - m$ would begin with n base-1 blocks on TOP and m on BOT (N.B., this is not the way base-1 block subtraction is ordinarily posed in the classroom).[5] There are two hands, LH and RH, which can perform two kinds of actions—namely, picking up one block (PICK/FROM) or putting down a block being held (PUT/ONTO). The primitive predicate EQUAL takes two piles of blocks and says whether they designate the same number. EQUAL is not executable and cannot appear in the final plan.

The foregoing constraints describe the mathematical goals of the procedure, the objects it works with, and the physical manifold within which it operates. The mathematical content of subtraction is expressed in constraints 1 and 2: TOP minus BOT is TOP whenever BOT is empty of blocks, but any changes in the number of blocks on BOT must be echoed by an equal change in the contents of TOP. The objects the procedure manipulates are base-1 blocks. Because these are very simple, constraint 3 suffices to describe them. (By convention, a lowercase letter stands for an arbitrary block, whereas an uppercase letter stands for an arbitrary place mat or hand.) The remaining constraints define the physical manifold within which the procedure will operate. Constraints 4 and 5 ensure that blocks are *conserved* by the actions PICK/FROM and PUT/ONTO. Constraints 6, 7, and 8 describe how the hands that manipulate the blocks work. A complete description of the workspace would require several more constraints, but these will do for purposes of illustration.[6]

[5] Dienes Block subtraction and other block subtraction procedures are usually taught using oral or written presentations of the problems. Thus, to solve $n - m$, the first step is to translate n into blocks, using some "bank" as a source of blocks. Next, one translates m into blocks, but *uses the first pile as the source*. When one is finished translating, the first pile contains $n - m$ blocks. This procedure for doing block subtraction is so dissimilar to written subtraction that we have avoided using it in this paper.

The constraints describe *domain-dependent* knowledge. If the procedure's goals or implementation environment change, then the constraints must be changed to reflect this. For example, if one used Dienes Blocks instead of base-1 blocks, then constraint 3 would be replaced by a new constraint, namely:

3′. a is EQUAL to b if and only if SHAPE(a) = SHAPE(b).

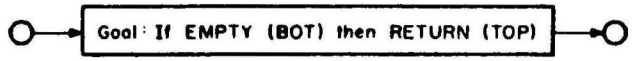
If one wished to plan an addition procedure instead of a subtraction procedure, then constraint 2 would become:

2′. The increase in TOP must EQUAL the decrease in BOT.

Heuristics are presupposed to be *domain-independent* knowledge. They represent commonsense planning knowledge, such as: "When you need to accomplish two things, and it doesn't matter which comes first, then pick one arbitrarily, do it first, then the other." We include this distinction between constraints and heuristics only because it is traditional; nothing in our theory turns on this distinction.

Planning a Base-1 Subtraction Procedure

The planning of the base-1 subtraction procedure involved 12 steps. Each step is an application of a constraint or a planning heuristic. The planning begins with a flowchart initialized to the constraint that is marked as the "goal" of subtraction.



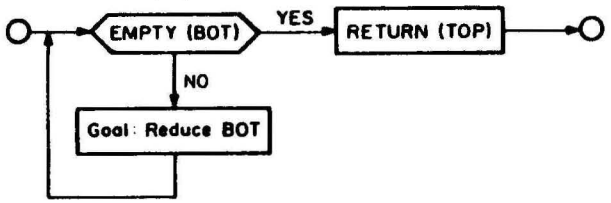
Planning proceeds by progressive refinement of goals to subgoals, or by checking the current plan against the constraints. (N.B., Because we are only interested in having a correct planning net for a procedure, not in *finding* one, we are going to ignore a few of the subtle issues.)[7]

[6] In formulating constraints, it is very important to put as little into each constraint as possible. For example, we could have replaced constraint 2 by "decrementing BOT by 1 must be echoed by decrementing TOP by 1." Although adequate for base-1 subtraction, this is not the most general statement of the constraint, and, indeed, this constraint would have to be replaced to handle Dienes Block subtraction. The basic idea is to split the declarative description of the world and the goal as finely as possible, so that small variations on the procedure can be modeled by replacement of one constraint among many small ones, rather than modification of one clause of a large, special-purpose constraint.

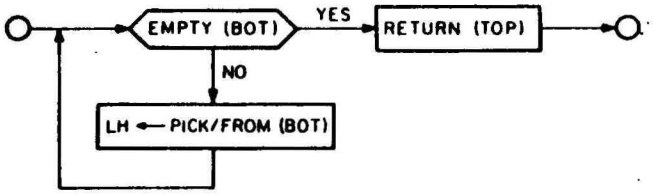
[7] We will gloss over a number of very difficult issues in the presentation of the planning steps. For instance, why was the TABLE chosen in Step 5 as the location for emptying the LH? How did we

Step 1: At the outset, the "implication-reduction" planning heuristic that reduces an implication ($A \supset B$) to a sequence of subgoals (A, B) can be applied. The second subgoal in this case is a primitive of the workspace. So the output of Step 1 is a plan with just one subgoal:

Step 2: A venerable planning heuristic, traditionally called "hill climbing" (Newell & Simon, 1972), reduces the goal to a loop. The loop test sees if the goal has been achieved, and if not, it takes a step "up the hill," so to speak.

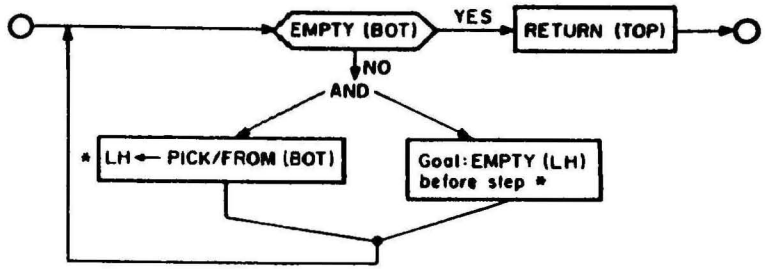


Step 3: The goal matches part of constraint 4—the definition of PICK/FROM. So the constraint is applied, and the plan is now fully reduced to primitive actions:

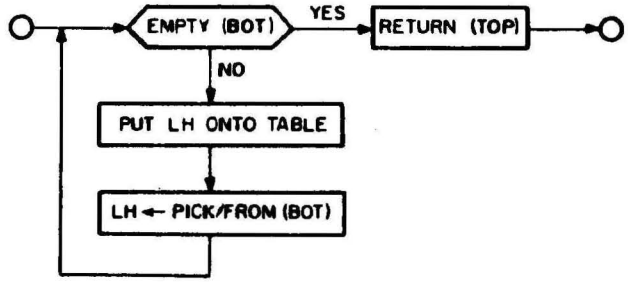


Step 4: Execution of this plan reveals a violation of constraint 6: The left hand must be empty before one can pick something up. So a new goal is created:

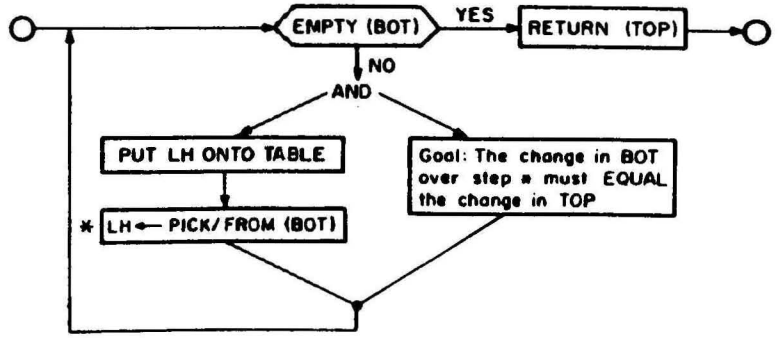
know not to empty it on TOP or BOT? Only the successful reasoning will be presented—the alternatives that didn't work aren't mentioned. Most of the research in planning for robotics has gone into improving the search for correct plans by recognizing unworkable alternatives and recovering from them gracefully. All these difficult questions involving search can be ignored, because we are not interested in automating the discovery of planning nets.



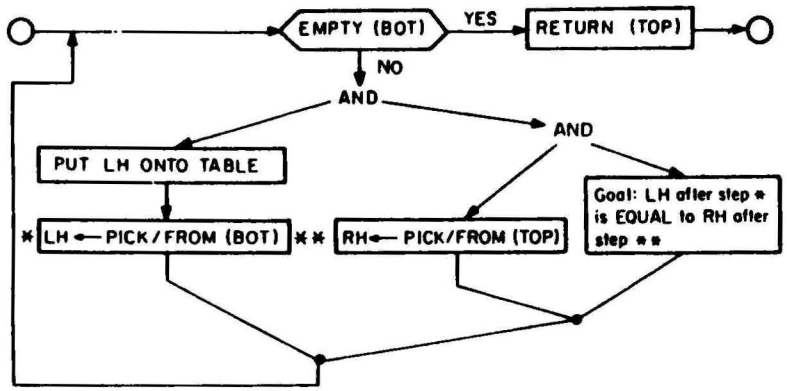
Step 5: This goal is quickly dismissed by applying constraint 7—part of the definition of PUT/ONTO. The left hand is now emptied before use.



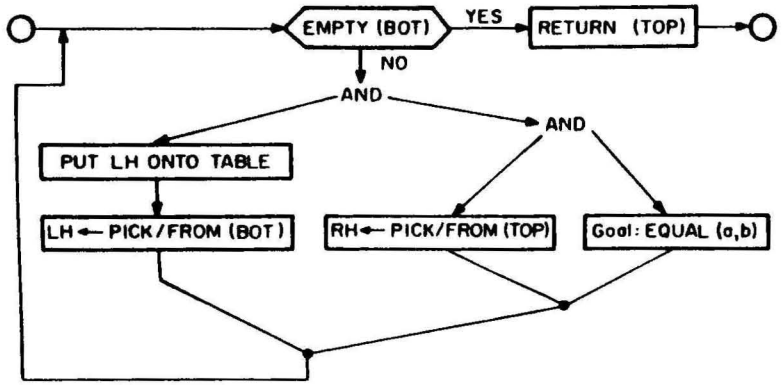
Step 6: Execution of this plan uncovers a violation of constraint 2. Because the bottom place mat is not empty when PICK/FROM is executed, one knows from constraint 8 that the left hand comes to hold exactly one block. Via constraint 4, one infers that the bottom place mat has its contents decreased by PICK/FROM. But there is no way to show that the TOP place mat undergoes an equal change. So constraint 2 is violated, and a new goal must be created. The goal says that there must be a change in TOP to equal the change in BOT.



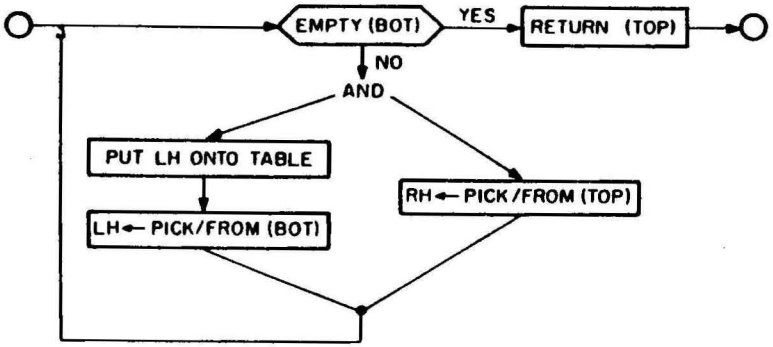
Step 7: Part of this goal matches constraint 4, the definition of PICK/FROM. A new picking-up action is instantiated for the top place mat. This reduces the goal of equal changes to the goal of equal contents of the left and right hands.



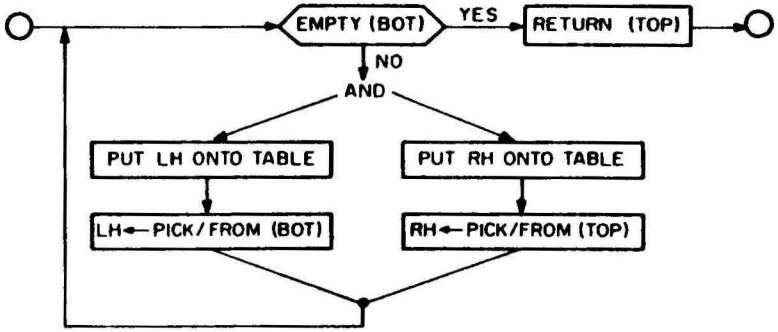
Step 8: Constraint 8 can apply twice now, once per hand. It says that only one block is picked up by PICK/FROM. Thus, the goal of EQUAL contents is replaced by equality of two arbitrarily chosen blocks.



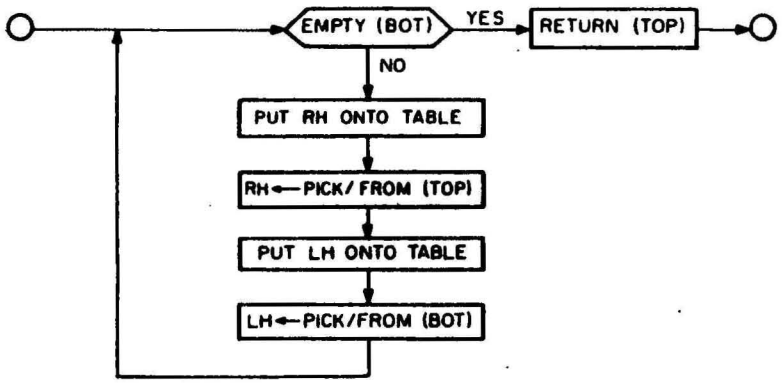
Step 9: Of course, this new goal is trivially satisfied by constraint 3—all blocks count the same in the base-1 number system. So the goal is simply removed from the plan.



Steps 10 and 11: Execution reveals that constraint 6 is violated again, this time by the right hand. So it must be emptied before use as well, in the same two-step fashion as Steps 4 and 5.



Step 12: A planning heuristic, call it "conjunction reduction," removes the conjunction AND. The AND node is for conjoining subgoals. It makes no statements about which subgoal to achieve first. In this case, it doesn't matter how the subgoals are ordered since they turn out to be independent. So the rule arbitrarily chooses the following ordering:



This is the final plan. Every step is a primitive, and all the constraints check out. The planning for base-1 subtraction is complete. The final plan is exactly the flowchart representation of the surface structure of the procedure.

Planning Nets

Planning nets are directed graphs. The nodes of the net represent plans, and the links represent planning inferences. That is, each node of the net stands for a flowchart containing a mixture of primitive actions and subgoals to be expanded. Two nodes are linked only if the application of some constraint or heuristic to one plan results in the other plan. The link is labeled with the planning rule that causes the change.

Sacerdoti developed a very similar structure to aid in automated task planning and monitoring in robotics. It is remarkable that we have found it useful for our research on procedural semantics, as has Greeno for his research on modeling the counting behavior of children (Greeno, Gelman, & Riley, 1978). However, we are faced with a clash in nomenclature. Sacerdoti calls these sorts of structures "procedural nets." We prefer to call them "planning nets," because their content has more to do with the planning of a procedure than with the procedure itself.

Planning Nets Are Partial Orders. In fact, planning nets are generally not sequences as the chronological presentation of the previous subsection might lead one to believe. Often, two planning inferences can be applied in either order. For example, step 6 could have preceded steps 4 and 5. To represent this independence, we allow the net to be a *partial order*.

Figure 18.2 shows the planning net for base-1 subtraction. In addition to the names of the planning rules, the steps have been labeled with the step numbers used in the previous subsection. The split at steps 4 and 6 occurs because

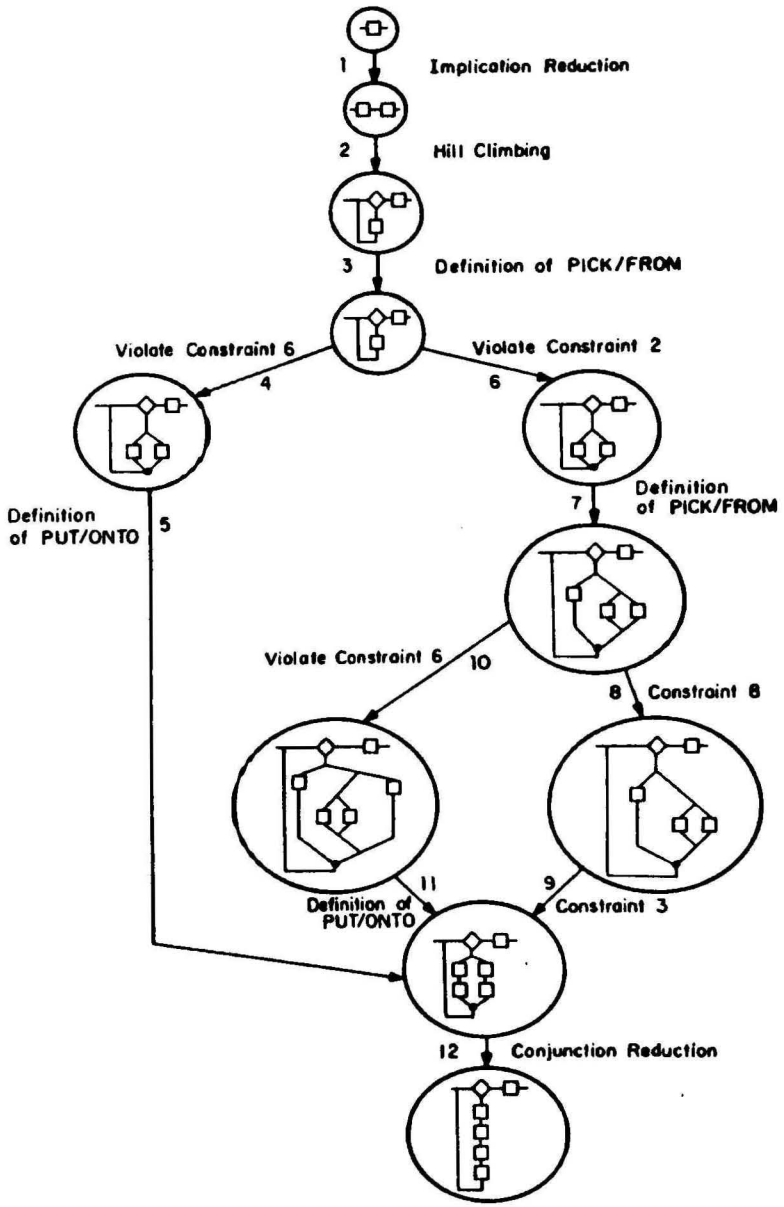


FIG. 18.2.

constraints 2 and 6 can be fixed independently. The other split shows that constraint 6, applied this time to the right hand, can be fixed independently of the subgoal reduction due to constraint 8.

Planning Nets Are a Complete Representation. The previous section may have left the impression that planning knowledge must be represented in three parts: the constraints, the planning steps, and the ultimate surface structure; and that planning serves as a transformation of the constraints into the surface structure. Although this is not a bad way to think of planning, it is unnecessarily redundant. The planning nets alone capture all three kinds of information. The constraints that are relevant to the procedure are exactly those constraints that appear as edge labels. Similarly for the heuristics. The surface structure is the contents of the bottom node, the final plan. So, planning nets are a complete representation of the design of a procedure.

Planning Net mp-Morphisms Formalize Procedural Analogies

To formalize procedural analogies, one merely applies the definition of "match" for directed graphs that was given in a previous section. That is, a procedural analogy is formalized as a graph-theoretic mp-morphism between the planning nets of the two procedures. We illustrate this definition with an example.

Figure 18.3 shows the planning net for a "big-pile" Dienes Block subtraction procedure. This procedure has the same sort of pairing-off action as the base-1 procedure discussed earlier, but it represents a number as a big pile of Dienes Blocks. Although space does not permit labeling the links in the planning net with their planning inferences, the step numbers should be sufficient to describe the match with the planning net of base-1 subtraction, which appears in Fig. 18.2. Step 9 of Fig. 18.2 is replaced in Fig. 18.3 by a subgraph consisting of steps 9.0 through 9.7. So all the links of Fig. 18.2 match the correspondingly numbered links in Fig. 18.3 except for link 9. The reason why link 9 can't be matched is simple: It is the application of the constraint that makes base-1 blocks all count the same—namely, constraint 3. In Dienes Blocks, all blocks do not count the same. Only if they are the same size do they designate the same number. What the subgraph of steps 9.0 through 9.7 is doing is planning out a way to get blocks that aren't the same size to be the same size by doing the appropriate trading. In fact, the planning leads off in step 9.0 by noticing a violation of the constraint 3', which says: "Only blocks that are the same size count the same."

The mp-morphism of the two planning nets results in the following intersection and difference subgraphs (calling the Dienes Block procedure A, and the base-1 procedure B):

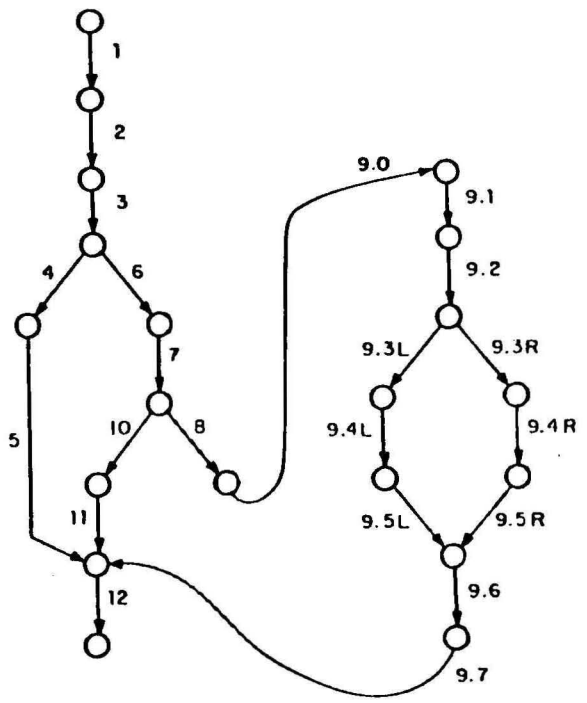


FIG. 18.3.

$A \cap B$ is almost the whole planning net for base-1 subtraction except the link for step 9.

$A - B$ is the subgraph that replaces step 9, whose steps are labeled 9.0, 9.1, and so on.

$B - A$ is just step 9 of the base-1 planning net.

The $A - B$ subgraph is almost the same size as the intersection subgraph, indicating that the closeness metric would probably give the analogy a rating of "moderate," which corresponds with the intuition nicely.

Difference Generators Are Used To Predict Closeness

As we hinted earlier, it is not always the case that the predictions based on the relative sizes of the intersection and difference subgraphs correspond so nicely

with the intuition. However, in those cases, the problem has been immediately apparent and was fixed, utilizing the fact that planning nets are *partial orders*.

To illustrate the problem, a new analogy is introduced and compared to the one described in the previous subsection. Whereas the earlier example was, intuitively, a moderately close analogy, this new analogy is quite a bit closer still. However, the simple view of the closeness metric as corresponding to the relative sizes of the intersection and difference subgraphs leads to the false prediction that the old analogy is actually closer than the new one.

Suppose we compare big-pile Dienes Block subtraction to sorted Dienes Block subtraction, an analogy that earlier provided a counterexample. For convenience, let us attach some letters to these procedures and the ones used in the earlier analogy:

A: base-1 subtraction

B: big-pile Dienes Block subtraction

C: sorted Dienes Block subtraction

The *BC* analogy is intuitively rather close. However, when the planning nets are compared, we find a huge subgraph of *C* that isn't matched—namely, all the design that has to do with maintaining the sort. Indeed, this difference subgraph, $C - B$, is much larger than $B - A$ and $A - B$ together. Subgraph $B - C$ is also quite large. Hence, even though $B \cap A$ is somewhat smaller than $B \cap C$, any reasonable metric would predict that analogy *AB* should be closer than analogy *BC*, contrary to the intuition that big-pile Dienes Block subtraction is more similar to sorted Dienes Block subtraction than to base-1 block subtraction. There is a mismatch between predictions of the theory and judgments of closeness.

But closer examination of subgraph $C - B$ reveals it has only one entering link, just like link 9.0 of Fig. 18.2. This link is labeled "Violates Constraint 11: keep blocks sorted by size." In other words, it appears that one plan inference is *causing* all the others. We can capture this notion of causation by utilizing the *topology* of planning nets.

As already discussed, planning nets are partial orders. Any subgraph of a partial order is also a partial order. In particular, the difference subgraphs are always partial orders. Any partial order has a unique set of *minimal elements*. This set is the smallest set of links that dominate all the other links in the subgraph. These mathematical facts ensure that the following terms are well defined:

> Where *X* and *Y* are any two planning nets, let $d(X - Y)$ be the links that are the minimal elements of the difference subgraph $X - Y$, and let $d(Y - X)$ be the links that are the minimal elements of $Y - X$. Call these two sets the *difference generators* of mp-morphism *XY*.

Difference generators are a formal representation of what is causing the difference between two procedures. Intuitively, what the difference generators of mp-morphism represent are the *crucial ideas* that separate the two procedures. All the other differences between the two procedures stem from these few crucial ones.

To illustrate this notion of "crucial ideas," take the analogy between base-1 and big-pile Dienes Blocks, which we were calling analogy *AB* in the previous section. $d(B - A)$ is a graph with just one link, labeled "Step 9: Constraint 3—all blocks are EQUAL." $d(A - B)$ is a link labeled "Step 9.0: Constraint 3'—two blocks are EQUAL if and only if they have the same SHAPE." Replacing constraint 3 by constraint 3' is about as clear a statement of the difference between base-1 blocks and Dienes Blocks as one can hope to make.

Because difference generators capture the distinctions between procedures so succinctly, they seem highly appropriate as the inputs (or arguments) to the closeness metric. They are decoupled from the unimportant details that fill flowcharts, procedural nets, and planning nets—details that obscure the essence of analogy by inflating difference subgraphs with derived, less meaningful structure. Indeed, the comparison of analogy *AB* to analogy *BC* (i.e., the big-pile vs. sorted analogy) now agrees with intuition: All four difference generators—namely, $d(A - B)$, $d(B - A)$, $d(B - C)$, and $d(C - B)$—are about one link big. On the other hand, the intersection subgraphs are as before, with $A \cap B$ being smaller than $B \cap C$. Because the difference generators are about the same size, the intersection sets are more important in the closeness metric. Hence, a reasonable metric would report that *BC* is closer than *AB*, which corresponds with the intuition that big-pile Dienes Blocks subtraction is closer to sorted Dienes Block subtraction than to base-1 blocks subtraction. At last, we seem to have found a level of abstraction for procedures where intuitions of closeness correspond to the relative sizes of the inputs to the closeness metric.

Discussion

The main point of this section has been that planning nets provide a basis for a theory of analogy that can predict the judgments of experts on the closeness of analogies between procedures. Moreover, all the aspects of the theory have very natural, almost elegant sources. The deep structure used came naturally from Sacerdoti's work in robotics; mp-morphisms are a general-purpose concept; and the notion of difference generators came naturally from the topology of planning nets.

We have always been struck by how much of the design of a procedure like subtraction is governed by the design of the representation of the objects manipulated by the procedure (e.g., the place-value number system). In fact, many of the actions in any of the elementary arithmetic procedures concern not the mathematical operation per se but rather *how the object representations are*

manipulated. This impression is reinforced by experience in computer programming, which is often a constant interplay between the design of the object (i.e., data) representation and the code, even at the highest levels. Anyone who has tried to understand a program that he or she did not write can vouch for the importance of understanding the data representation. In the process of judging the closeness of an analogy, a popular strategy is first to look at each procedure's object representation, and then to build the understanding of the overall analogy on the basis of the analogy between object representations. In short, it appears to us that a large portion of the "understanding" of a procedure consists of an understanding of the implications of the procedure's object representation.

This view of procedural understanding is entirely consistent with the planning-net formalism. The constraints and heuristics that appear in the net represent, in some sense, the *essence* of the procedure. If object representations were unimportant, then none of the planning inferences would be "about" the object representation. But, in fact, many planning inferences do deal with the object representation. Even in the foregoing base-1 blocks procedure, with its extremely simple object representation, we find constraint 3 addressed solely to the object representation. In more complex procedures, using Dienes Blocks or written numerals, an even larger portion of the constraints concern the object representation. In short, although planning nets abstract out the less important aspects of a procedure, they leave behind the design of the object representation, which is quite compatible with the view that as a representation of "understanding" of procedures, a fair portion of the design should model the "understanding" of the object representation.

We have not discussed the exact definition of the closeness metric, even though some definition would be necessary to verify methodically the correlations we have claimed. There are many difficulties and fine points involved in determining such a definition. In particular, it is plausible that the weight of some planning inferences is quite close to zero. We have in mind the commonsense heuristics, such as implication reduction, that play an almost invisible role in the planning. Also, some planning rules are applied more than once in a planning net; one may perhaps wish to avoid giving such rules an inappropriate prominence by only counting their first occurrence in the difference generators or the intersection sets. These are just two of the many points one would have to consider in defining a closeness metric.

The reader has no doubt noticed the incredible amount of work that goes into analyzing a procedure in terms of its planning. First one constructs the flowchart, then the constraints and a sequential plan for the flowchart, and last calculates the planning net by noting which planning inferences are not ordered with respect to each other. This large amount of work leaves much room for error on the part of the theorists. However, each level of abstraction is well defined and can be checked for consistency by a computer. Thus, one next step is to build a computer system of utilities to aid in the analysis of procedures. However, there is a certain amount of intuition that goes into some parts of the analysis, notably the formulation of a set of constraints, that we doubt could ever be successfully mechanized.

ANALOGIES AND TELEOLOGIC SEMANTICS IN EDUCATIONAL RESEARCH

In this section we consider some of the issues involved in explaining (or teaching) the knowledge we discussed in the first previous section—teleologic semantics. Briefly, teleologic semantics is the kind of knowledge that concerns the purpose of each part of the procedure, as well as the motivation behind the set of constraints that defines the particular representation for the objects. In particular, we consider how an individual piece of teleology can be explained, and how such individual explanations can be combined into an integrated explanation.

The section closes with a discussion of some issues involved in microworld-based curricula. These issues turn out to be intimately related to those involved in teaching teleologic semantics.

Local Explanations: Manifestation and Motivation

An important property of the planning-net formalization is that there is a *natural* notion of how to explain a small piece of a procedure's teleologic semantics. By "piece" we mean a constraint (or a small set of constraints) that is used in the planning net. To "explain" it, one uses a *minimal contrasting pair* of procedures—one with the constraint, and one without it—that compute the same "operation" as the given target procedure. In other words, we use analogies to illustrate constraints. We believe that using a concrete surface structure illustration for each deep structure concept is a very important explanatory technique that naturally falls out of this development. For example, this method frees us from having to explain the planning formalism to the student—a task potentially more difficult than teaching the procedure itself.

More formally, to illustrate some given constraint(s), one uses *two analogous procedures such that one of the difference generators of the mp-morphism between them is exactly the given constraint(s)*. If the pair of procedures forms a minimal contrasting pair, then the mp-morphism constituting the analogy is *elementary*.

Of course, this technique works just as well for explaining heuristics. However, heuristics are often such commonsense knowledge that an explanation of them is unnecessary. So we call the planning inferences to be explained "constraints," avoiding the cumbersome phrase "constraints or heuristics." Also,

our terminology reflects the fact that it is often possible to provide a minimal contrasting pair for each constraint individually (this observation is discussed later). So we use "constraint" in place of "a small set of constraints."

An important realization is that minimal contrasting pairs can be used in two different ways in an explanation. They can be used to show how the constraint is *manifested* on the surface, and they can also be used to *motivate* the inclusion of the constraint in the ultimate design of the procedure. Probably the best way to illustrate the differences between these two uses is with an example.

Explaining the Canonicity Constraint. The particular constraint that is used in this example is one of the most subtle and influential in arithmetic—namely, the *canonicity constraint*. To show how the planning-net representation can aid in explaining procedures, the constraint is presented as the "answer" to a nontrivial teleologic question.

What is the purpose of carrying? More specifically, if the problem is 52 + 49, why bother to carry 10? Why not leave 11 in the units place? It is not because there is no symbol for the "digit" 11—we could invent one if we wanted. In Dienes Block addition, the question is even clearer. Why not leave the answer in the form of 9 longs and 11 units? Why bother carrying?

The answer is that carrying maintains the *canonicity* of the representation of numbers. A canonical representation puts the representational objects in one-to-one correspondence with the real objects they represent. The Hindu-Arabic representation of numbers is canonical because there is a unique, distinct numeral for each number. Dienes Blocks are not necessarily a canonical representation, since most numbers can be represented several ways. For instance, 11 can be represented as a long and a unit, or as 11 units. The purpose of carrying is to canonicalize the sum by making sure that there are no more than nine blocks of any given shape. In other words, carrying is the *manifestation* of the canonicity constraint.

But suppose that the questioner rejoins by asking what the purpose of the canonicity constraint is. The answer involves another arithmetic subprocedure—comparison.

It is much more efficient to find out which numeral represents a given large number if the representation is canonical. Let us use a Dienes Blocks comparison procedure to illustrate the gain in efficiency. In a noncanonical representation, the comparison procedure must compare all the piles, because a very large pile of small blocks can make up for a deficit of larger blocks. In a canonical representation, the comparison procedure needn't check all the piles. If it finds that one numeral has more flats than the other numeral, then it needn't compare the longs or units: even if the other numeral has the maximum number of longs and units allowed—namely, nine each—the first numeral will still represent the larger number. Imposing the canonicity constraint makes the comparison procedure much more efficient, because it allows the procedure to stop earlier. But the canonicity constraint is a constraint on the representation of numbers, and so all arithmetic procedures must obey it. Even though the constraint makes part of the addition procedure somewhat less efficient, it makes comparison so much more efficient that it is worth having. This appeal to efficiency is the ultimate end point in the explanation of the *motivation* for carrying and the canonicity constraint.

In this miniexplanation of carrying, we have seen two important facets of teleologic knowledge. In the addition procedure, the canonicity constraint was *manifested* as a carry subprocedure. But the *motivation* for adopting the constraint lay in another procedure, comparison. Each of these two facets, which we now call *local* explanations because they explain just one constraint, was illustrated with a minimal contrasting pair of procedures. One member of the pair was a fully operational version of the procedure that lacked the constraint being discussed, whereas the other member adopted the constraint. But the manifestation part of the explanation involved a minimal contrasting pair that was different from the pair used to motivate the constraint (i.e., addition vs. comparison). As discussed later, it is preferable to have a pair of analogous procedures that illustrate both the manifestation and the motivation of teleologic concepts, but this is not always possible.

It is our belief that the concreteness of this minimal contrasting-pair paradigm of explanation is of crucial importance in making teleologic semantics clear. The learner can *see* in very concrete terms how adopting a constraint affects the procedure. Winston showed that a similar example-based paradigm was sufficient to teach the abstract concepts necessary to recognize toy block constructions, such as an arch (Winston, 1975, 1978).

In fact, many minimal contrasting pairs that manifest the given constraint are available, depending on which of the remaining constraints are adopted. If all the constraints of a given target procedure are adopted, then one member of the pair is the target procedure itself. Otherwise, the contrast is exhibited across a pair of *model* procedures that still satisfy the mathematical constraints of the target procedure. Using model procedures often highlights the contrast, making it much easier to see the constraint under discussion. Such was the case with the canonicity constraint, where Dienes Blocks allowed us to use noncanonical numbers without inventing new digit symbols.

However, model procedures must be used with some care, as the following example illustrates.

The Impact of Efficiency Metrics on "Loop Jamming." Consider the difference between the standard carry subprocedure and the two-pass version described in the introduction, where carrying was deferred while all the columns were added, then performed on a second pass over the columns. This difference is a constraint that was called *loop jamming*, after the compiler optimization technique of the same name that weaves two loops into one (Allen & Cocke, 1972).

One cannot use Dienes Blocks procedures to motivate loop jamming, because exactly the same number of hand motions, fact-table lookups, and so on are required by each procedure. So, Dienes Blocks are an inappropriate model domain for discussing this constraint.

However, when implemented with written numerals, loop jamming does create a difference in efficiency.[8] The two-pass implementation of carrying requires more writing than the standard implementation. Thus written arithmetic turns out to be an appropriate domain for discussing the loop-jamming constraint.

The important point to notice about this example is that the choice of the model has some impact on the local explanation. In particular, a model that clearly displays the *manifestation* of the constraint in the procedure may not be able to demonstrate the *motivation* for the constraint. For example, because one doesn't have to worry about how to write the intermediate column sums that may be greater than 9 with Dienes Blocks, we can use them to implement both the one- and two-pass addition procedures and thus use them to illustrate the manifestation of loop jamming. Unfortunately, however, they cannot be used to motivate loop jamming, because the resulting procedure is no more efficient.

Another point to notice about the preceding example is the use of *efficiency metrics* in motivating design choices. An efficiency metric is some weighted sum of hand motions, fact-table lookups, table size, amount of paper used, and the like. The weighting of efficiency metrics is very important. For example, if reducing memory load is more desirable than decreasing the number of write operations, then the discussion of loop jamming ends with the opposite conclusion—that two-pass carrying is better than the standard subprocedure.[9] The two-pass version uses less short-term memory but more pencil lead. So exactly what efficiency metric is used greatly affects the local explanation. We do not look upon efficiency metrics as a regrettable new variable that must be tied down and parameterized with careful experimentation, but rather as a source of flexibility that can be used to tailor the teaching paradigm to the needs of particular students.

Principles for Sequencing Local Explanations

For moderately complex procedures, such as subtraction, the number of constraints can be high enough to cause problems of presentation. Our current best

[8] In the standard version of subtraction, where the carry loop is jammed together with the add-column loop, one must write $n + m$ digits, where n is the length of the longest addend and m is the number of carries required (it is assumed that one writes a 1 above the columns one carries into). In the two-pass version, one must write $n + 2m$ digits: One must remember from the first pass which columns are overflowing, and this requires m notes to oneself—say, in the form of writing a 1 above the overflowing column. The second m operations come from rewriting the answer digit of the columns that are carried into. There may be even more rewriting if the answer carried into is a 9.

[9] In the column carried into, the standard subprocedure requires adding three digits, one of which is, of course, the carried 1. But adding three digits requires remembering the sum of the first two digits while assessing the third digit. The two-pass subprocedure doesn't load memory this way, because the intermediate sum is written down instead.

estimate of the number of constraints of subtraction is 17. To explain this many constraints, each with its own manifestation and motivation, may seem a difficult task. However, with the planning net formalism, we can investigate how to sequence "optimally" a collection of "model" procedures; the first procedure (or "model") of the sequence would be a very, very simple version of the skill, and the last procedure of the sequence would be the target procedure. For example, in subtraction, the first procedure might be base-1 block subtraction and the last, standard written subtraction. But how should the intermediate models be sequenced?

Using the formalisms developed earlier, principles for sequencing local explanations can be stated precisely. Several such principles are stated next that we believe will lead to sequences that better enable assimilation of the overall teleology of a procedure from the explanations of its parts. Each one of them falls out quite naturally from the planning net formalism.

It is convenient in what follows to say that such sequences run from left to right—the target procedure is the procedure on the far right. This allows us to talk of the left and right procedures of a mp-morphism. Also, we speak of the left and right difference generators of an mp-morphism; if A is left of B, then $d(A - B)$ is the left difference generator.

Introduce Each Constraint. As we saw in the previous subsection, it is best to illustrate each constraint with a minimal contrasting pair of analogous procedures. This is probably the most important sequencing principle, that each constraint be illustrated individually. However, it is probably also true that it is better to introduce the constraint rather than take it away. This gives the sequence an air of progression toward the target procedure. Putting this principle formally, we have: Each constraint is the sole contents of the right difference generator of some mp-morphism in the sequence. That is,

Principle 1. For each constraint C in the target procedure's planning net, there exists i such that $d(P_i - P_{i-1}) = \{C\}$.

where the procedures are numbered from left to right (first to last).

Starting with a very simple procedure would, hopefully, tap a person's intuitive understanding. Then, since each of the analogies (mp-morphisms) is very close (or at worst, moderate; we are guaranteed only that one of the difference generators is a singleton set—namely, the constraint being introduced), it should be easy to transfer that understanding along, *augmenting* it only *slightly* as each new procedure is presented.

Only Introduce Target Procedure Constraints. Occasionally, it is necessary to "build" a left procedure to illustrate some constraint. This occurs when one cannot adjust the sequence so that the right procedure of some other constraint is this constraint's left procedure. In this case, one ends up with an adjacent pair of procedures that do not illustrate a constraint from the target procedure. Although the person (or computer) doing the explaining can mention that this analogy isn't so important, it would be better if the sequence didn't have such pairs. So another optimization principle to shoot for is:

Principle 2. For each i in the sequence, there exists a constraint C in the target procedure's planning net, such that $d(P_i - P_{i-1}) = \{C\}$.

Minimize Redundancy. One should not remove a constraint that has been introduced previously or introduce a constraint twice. Although one could argue that the redundancy of seeing the constraint illustrated in several different contexts (i.e., with different model procedures) serves to reinforce the local explanation, we are of the opinion that this would create confusion rather than dispel it, and in addition, it would create the impression that the sequence was meandering.

More formally, we propose that the sequence obey the following conditions:

Principle 3. For any $i \neq j$ $d(P_i - P_{i-1}) \cap d(P_j - P_{j-1}) = \phi$

Principle 4. For any $i \neq j$ $d(P_{i-1} - P_i) \cap d(P_{j-1} - P_j) = \phi$

Principle 5. For any ij $d(P_i - P_{i-1}) \cap d(P_{j-1} - P_j) = \phi$

The first condition advises one not to introduce a constraint twice, and the second condition advises one to avoid removing a constraint twice. The third condition says that once a constraint is introduced (the first term), it can never be taken out (the second term). Actually, it also says that once a constraint is removed, it shouldn't be reinserted, which is also a plausible condition to impose for aiding the cogency of the sequence.

Efficiency Should Increase Monotonically. We mentioned earlier that a minimal contrasting pair for a constraint does not necessarily show an increase in efficiency. That is, all ways of manifesting a constraint do not necessarily motivate it as well. One condition on a sequence is that the model procedures be chosen and sequenced so that efficiency always increases as the target constraints are adopted. That is,

Principle 6. For all i, P_i is more efficient than P_{i-1}.

Because there are many minimal contrasting pairs that manifest a constraint, it is usually not difficult to find some pair that motivates it as well, but putting that pair into a sequence with the other constraint's pairs can be somewhat difficult. We know of only one constraint for addition or subtraction—namely the canonicity constraint, where the motivation pair *must* be distinct from the manifestation pair. This is inevitable because canonicity is basically designed to improve the efficiency of comparison, not the other arithmetic operations. Thus, if one were only interested in a sequence of addition procedures or subtraction procedures, then the pair for the canonicity constraint would necessary violate this sequence principle. However, with this one exception, it has been easy to fine *some* minimal contrasting pair that serves both to manifest and motivate a constraint for subtraction.

However, putting such pairs into a sequence requires some care. Switching the order of two constraints in a sequence often alters the relative efficiency of the minimal contrasting pair of procedures that manifest the unit. Under one ordering, both constraints might improve efficiency. But under the reverse order, adopting one of the units may result in no increase in efficiency or even a decrease in efficiency. This might seem strange, so let us pause a moment for an example.

Consider ordering the canonicity constraint versus the constraint that Dienes Blocks be kept sorted by size. First, suppose that the canonicity constraint precedes the sort-by-size constraint in the sequence. Under this ordering, the efficiency increases between each procedure; imposing the canonicity constraint forces the procedure to search through the big pile of Dienes Blocks to check that there are no more than 10 blocks of any given shape. Hence, adopting the sort-by-size constraint greatly improves efficiency by eliminating rummaging around through the big pile in favor of simply counting up the number of blocks in each of the small piles.

Now suppose the order in the sequence were reversed and sort-by-size were imposed before canonicity. The minimal contrasting pair for sort-by-size consists of: (1) adding two big piles of Dienes Blocks together by simply forming the union versus (2) adding each of the small piles together in a series of separate union operations. The introduction of the constraint actually decreases the efficiency of addition. Because no carrying is required (canonicity not being imposed yet), there is no use in the separation by size. Maintaining the constraint creates extra work with no reward. So modifying the order of two constraints in the sequence can have an impact on the ability to motivate them.

Although it may be a difficult condition to achieve, if a manifestation-based sequence has monotonically increasing efficiency, the viewer can see with no additional examples not only *what* each constraint is but also *why* it exists (i.e., what good it is).

Telescoping Sequences. Occasionally, one finds mp-morphisms that introduce a constraint but don't need to remove any constraints. The canonicity constraint can be illustrated with an mp-morphism whose left difference subgraph is null (for addition, one could use the two-pass addition procedure de-

scribed in the introduction as the right-hand procedure . . . the first pass of it for the left procedure). That is, the mp-morphism is *total* with respect to the left planning net. It seems plausible that mp-morphisms that never removed constraints would create a very strong sense of progression toward a target procedure. Such sequences are characterized by the condition:

Principle 7. For any i, $d(P_{i+1} - P_i) = \phi$

A Space of mp-Morphisms

Needless to say, it will rarely be possible for a sequence to satisfy all the sequencing principles we have mentioned. Indeed, we may only be able to satisfy some principles along part of its length and different principles along another part. We need some way to study the relative contributions of the various principles to ease of explanation.

Ultimately, we would like to develop a representation of all principled sequences to a given target procedure. These sequences could be represented in an economical way by a directed graph whose nodes would represent planning nets. There would be a link from node A to node B only if they appeared as an adjacent pair in some sequence that was considered a plausible explanation sequence, perhaps because it met some minimum number of the principles listed earlier. (In particular, one might include all (known) minimal contrasting pairs for the target constraints; this would correspond to using principle number 1 as a threshold for inclusion in the space.) This directed graph has the property that any sequence from a "most primitive version" node to the "target" node would be a possible sequence for explaining the teleology of the target procedure. We tend to think of this graph as a *space* of mp-morphisms.

One clear problem that could be attacked with such a space is improving on the *naturalness* of teleologic explanations. Presenting the 17 or so mp-morphisms (or procedural models) for place-value subtraction is bound to be very confusing unless they can somehow be aligned along the individual's own cognitive structures (see the Appendix for a detailed example of one such chain of models). We have already mentioned seven principles that probably contribute to better comprehension of such explanations. Each of these principles would be incorporated into the space, perhaps as annotations on the basic partial order. Hopefully, experience and experiment will lead to the discovery of other factors that improve the naturalness of teleologic explanations.

Using the mp-Morphisms Space in Microworld-Based Curricula

In a microworld-based curriculum, the student explores a rich environment, hopefully inventing something analogous to the target skills (Papert, 1978; Fischer, Brown, & Burton, 1978). For example, a student might be given Dienes Blocks and a puzzle that requires using multidigit arithmetic to solve it. Actually, how students are motivated to do the arithmetic is not an issue here. The point is that students are not given the sequence of actions that implement arithmetic for the given representation of numbers. Instead, they must invent it themselves.

Tracking a Student's Discovery Process. The mp-morphisms space could be quite useful as a way to "track" a student's discovery process. The basic idea is that an observer (possibly a computer) analyzes the procedures that the student invents in terms of planning nets. The nodes in the space that correspond to the plans of these procedures are marked. The student's progress is then expressed as the shortest sequence along the constraints that connect the marked nodes. This provides a strong hypothesis concerning what the student has learned during the discovery process.

Such a tracking study would provide an empirical way to verify conjectures about "natural" sequences for teleologic explanations. That is, observing that students generally followed sequences that increase the efficiency of the procedure would support the conjecture that monotonically increasing efficiency is important for cogent, natural explanations.

Sequencing Microworlds. A persistent problem in microworld-based curricula is how to sequence the microworlds so as to maximize the cumulation of intuitions built up while exploring the microworld and enable them to be transferred to the target procedure. One ready answer is provided by the space of mp-morphism sequences, assuming it has been annotated to show which sequences are most natural.

Sequencing microworlds obviously imposes an order on the traversal of the nodes in the mp-morphism space. One can't move from a Dienes Block procedure to an abacus procedure's node until one leaves the Dienes Block microworld and enters the abacus microworld. So the most natural sequence of microworlds is the one that enables traversal of the most natural sequences through the constraint space. Let us illustrate this conjecture with an example.

Suppose one tried to teach addition with the following sequence of microworlds:

base-1 blocks, the abacus, Dienes Blocks, written numbers

One would expect the students to become frustrated when they find that the teleology associated with place-value encoding of numbers, which they laboriously invented for the abacus, is obviated by the shape-value encoding of Dienes Blocks. And when they find they must resurrect this place-value notion to move from Dienes Blocks to written numbers, one would expect them to become disgruntled or, worse yet, to apply "teacher psychology" and guess that place

value couldn't possibly be part of the design because "we already had that." In comparison, reordering the sequence to be

base-1 blocks, Dienes Blocks, the abacus, written numbers

allows invention of the notion of place-value just once, in transition from Dienes Blocks to the abacus, and then maintenance of the notion throughout the abacus microworld and on into the written numbers.

These ordering results could be predicted on the basis of one of the naturalness principles mentioned earlier—namely, that constraints ought to *accumulate* along the sequence. They should be added once and never removed. In the first sequence of microworlds, there is no sequence of procedures that can avoid adding the constraints that express place-value encoding during the first transition and dropping some of them during the second transition.

What Is the Closest Possible Procedure in a Given Microworld to the Target Procedure? Just exactly how close to standard arithmetic procedures can procedures built around a particular representation of numbers, say Dienes Blocks, be made to be? Can a Dienes Block procedure be devised that is totally isomorphic to a standard written procedure? This is a question of interest to educators. For example, it bears on the question of just how much a child can learn about standard arithmetic by inventing a good arithmetic procedure in a given microworld, such as Dienes Blocks. This in turn bears on the question of how many microworlds, and which ones, are necessary to allow the student to easily converge upon the target skill. With a formal theory of analogy between procedures, we can now precisely determine how close the best possible procedure defined over a given microworld can be to the target procedure.

Take any procedure that uses the given representation of numbers. Examine the difference generator of the analogy between it and the target procedure (e.g., written addition). If this set contains constraints that cannot be met because of the basic physics of the representation, then one cannot construct a model procedure that is isomorphic to the target procedure. An example should make this a little clearer.

A careful examination of the planning net has shown that it is impossible to construct a Dienes Block addition procedure whose analogy with written addition is perfect (i.e., an isomorphism). One design issue that is always present in Dienes Blocks involves the shape-value encoding that is the hallmark of Dienes Blocks. There is an encoding of the relationship between position and place value that is present in both written addition and sorted Dienes Block addition, but it is redundantly coded by the visual appearance of Dienes Blocks. If one got rid of this redundancy by evening out the sizes of the blocks, then they wouldn't be Dienes Blocks anymore. So the redundancy is inherent in the representation and will be part of the difference generator of the analogy to written addition no matter how clever one is about inventing Dienes Block addition procedures.

As a consequence, certain subtle *shifts in representation* that occur in the standard procedure for adding written numbers cannot be duplicated in any Dienes Block addition procedure.[10] This deficit gives some bite to the inherent incompleteness; the subtlety of these shifts makes them likely candidates for misunderstandings that Dienes Blocks are apparently helpless to prevent. This essential inadequacy can be directly diagnosed, if not predicted, using the theory of analogy between procedures.

In similar fashion, other microworlds can be evaluated. This evaluation is, however, quite constructive. Once the inherent mismatch with the target procedure has been identified, the gap can be filled by modifying the microworld, or by adding another microworld to the curriculum if desired.

In short, many of the same issues appear to be involved in the teaching teleology and discovery-based teaching. Planning nets seem to provide a formal tool for investigating this relationship further.

CONCLUSIONS

The major claim of this chapter is that planning nets provide useful formalisms for capturing the teleologic semantics of procedures. However, probably the most importnat thought to take away from this exposition is the importance and utility of using *planning knowledge* in the deep-structure analysis of procedures.

In contrast to other work on analogy, we have ignored the process of solving an analogy problem. Instead, we have concentrated on an intuitive determination of what representation most closely models the way experts conceive of procedures in order to understand analogies. This methodology has arrived at the same conclusion that was reached by a completely different method. In particular, our planning nets are very similar to Sacerdoti's "procedural nets" (Sacerdoti, 1977). Sacerdoti has shown his procedural nets to be a *sufficient* representation for designing procedures and indeed much better than other known representations. We have tried to show a similar representation to be a *sufficient* representation for judging the closeness of analogy and indeed much better than other known representations. In short, evidence is accumulating that planning net-like representations are good for many purposes. However, we should point out once again that neither Sacerdoti nor ourselves make any claims that the process of building a planning net, either for analogy or design, exactly models the human *process* of building a planning net.

---

[10]When one adds two large digits from a given column, one gets back a nondigit—for example, 14. The first shift in representation is to break this number down into units and 10s. Next, the units must be converted into a digit in the columns being added, whereas the 10s must be converted into an argument to the carry subprocedure. In Dienes Block addition, the second conversion is superfluous, because the result of the column addition is already scaled up to the value of the column, so to speak. That is, an add in the 10s column yields 140 in the form of 14 LONGS, not 14 UNITS.

Because teleologic knowledge is a part of a certain kind of expertise, one naturally wonders how it can be taught. Planning nets provide a precise framework for constructing explanations and curricula to explicate teleology. In particular, the formalism helps answer the question of how to sequence a set of "model" procedures with certain formal properties. Moreover, many of these same formal properties seem useful in discovery learning curricula.

Our last comment should undoubtedly be that this research is just beginning. There are many deficiencies and questions that must be addressed. Reliable empirical measurements of closeness and transferability must be made. The uncertainties in the uniqueness issue must be investigated. The general precision of the theory must be improved, and its inordinate amount of detail must be tamed, hopefully with the aid of a computer. In particular, we would like a complete, precise, mp-morphisms space for all five arithmetic operations. The limitations of the theory should be tested by exercising it on examples from other domains. In other words, this chapter is more a proposal to investigate a promising line of thought than a report on completed research.

APPENDIX: AN EXPLANATION OF THE TELEOLOGIC SEMANTICS OF SUBTRACTION

To give a feeling for how an explanation based on paths of minimal contrasting pairs of analogous procedures might go, an example of such a path is presented here. It begins with a base-1 subtraction model, passes through some Dienes Block subtraction procedures, and ends with the standard procedure for subtraction of written numerals. Although reading these rather abbreviated descriptions can have nothing like the impact of actually handling the blocks and doing the procedures, the power of this technique to explain teleologic semantics should nonetheless be apparent.

Throughout the path, there is a certain ambivalence about the particular material that is used in the representation of number. In fact, the primitives and constraints used to describe and implement procedures really can't differentiate real, wooden Dienes Blocks from, say, drawings of Dienes Blocks, as long as they are manipulated the same way. In fact, there is no particular point where adoption of the constraints of the target procedure (written subtraction) forces us off the counting table and onto paper; one can actually implement standard subtraction with cards bearing digits.

However, the material *does* make a difference to the efficiency metrics. In particular, some of the later constraints can only be motivated by assuming that erasing is more work than writing, which is true of paper but hard to emulate with manipulable materials.

We start with base-1 blocks because the mathematical semantics of this subtraction procedure are simple and concrete.

1. *Polynomial.* Base-1 numerals are rather bulky for representing large numbers. One solution to the block management problem is to let some counters stand for a fixed number of the unit counters. This is the *polynomial* constraint (3' in the text). The next procedure of this mp-morphism is a simple version of big-pile Dienes Block subtraction.
2. *Search Instead of Random Choice.* This model adds the notion that searching for two blocks of the same shape is more efficient than picking two blocks at random, then trading to make them the same shape.
3. *Chose Larger to Trade Down.* The idea here is to trade down the larger of the two blocks. If one picks an arbitrary block to trade down but not the unit block, then eventually one will be able to match their shapes, but it will often take more trading than always picking the larger one to trade down. This procedure requires memorization of which of two shapes stands for a larger multiplier.
4. *Search for Next Larger Before Trading.* When one can't find two blocks of equal shape, and instead has two blocks of unequal shape, then before trading down the larger one, replace it with a block that is the *next size larger* than the smaller block. If the search succeeds, one only has to trade down once. This plan step requires memorizing which shape is the next larger one than a given shape.
5. *Choose TOP to Trade Down.* This model is motivated by observing that when the block that is traded down comes from BOT (the bottom numeral), the subtraction as a whole takes more time than it would if the block had come from TOP (the numeral that is being subtracted from). When a block from BOT is traded down, the nine smaller blocks that are left over go back into BOT. So the main loop must run nine times more. If a block comes from TOP, the nine extras go back into TOP. If BOT runs out soon, they may never be touched. So trading down a block from TOP is more efficient than trading down a block from BOT.

   The goal of choosing TOP blocks creates a subgoal that the TOP block be larger than the BOT block. This subgoal is satisfied by a subgraph that is already a part of the left planning net—namely, the union of the subgraphs generated by models 2, 3, and 4. So the new part of the planning net underlying this procedure is just the part that satisfies the goal "choose BOT block" exclusive of the part that satisfies the subgoal.
6. *Canonicity.* This constraint was described earlier.
7. *Base Ten.* The canonicity constraint produces a trading pattern that is much easier to remember if all the multipliers are powers of 10 (or some other base). For example, in canonical American money, which is a

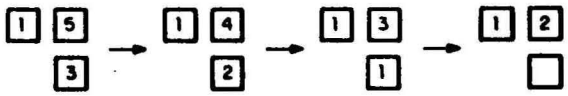
polynomial representation but not a base-10 representation of number, a citizen would canonicalize their pocket change by trading in five pennies for a nickel, two nickels for a dime, three dimes for a quarter and a nickel, and so forth.

8. *Sort by Power.* Canonicalization (= carrying) and decanonicalization (= borrowing) are somewhat easier if numerals are sorted so that all counters of a certain power are accessible at once. Dienes Blocks, as we observed them being used in schools, lacked this constraint. In fact, Dienes Blocks lack the canonical and base-10 constraints as well. However, teachers usually require their students to obey these two.
9. *Power Represented by Location Only.* Numerals must take up space, either on table tops or on paper. Once powers are sorted, location in space redundantly represents the power of a counter. In this mp-morphism, that redundancy is removed by making all coefficient tokens (i.e., "digits") look the same, regardless of the power. The abacus, for example, obeys this constraint. This allows one to represent much larger numbers, since one need not invent new token shapes when one needs to use a new, higher power. That is, one can make an abacus of arbitrary width, but Dienes Blocks, which are inherently unable to obey this constraint, are limited in practice to, at most, four powers.
10. *Zero.* To use location to represent power, a prearranged pattern of locations must be used. But such fixed patterns, like the abacus or columnar ruled paper, can't represent numbers that are larger than they have been designed to represent. Moreover, producing the patterns accurately is difficult to do freehand. A good solution to this problem is to represent power with relative locations, which amounts to using *zero* as a placeholder. A "relative-location abacus" could be built that lays out piles of beads in a line on the table; it would use a clear plastic bead as a placeholder and piles of colored beads as nonzero "digits."
11. *Alignment.* In setting up the subtraction problem, one insists that the numerals be aligned so that digits of the same power are in the same column. This reduces the effort necessary to locate the digits of matching power when subtracting.
12. *Noncountable Coefficients.* It is quicker to arrange counters on a table or write coefficients symbols on paper if the number of counters or strokes is small. This motivates replacing countable coefficients with symbolic ones (e.g., digits). However, with symbolic coefficients, the PICK/FROM operation is radically altered. It is no longer possible to decrement a coefficient by picking up a piece of it (i.e., picking up a block or erasing a hash mark). Instead, a decrementation table must be memorized. That is, one must be able to count backwards from 20.

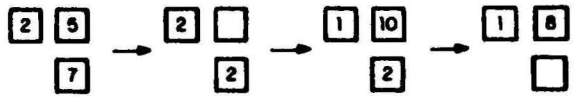
There is no particular point where the target constraints force us off the counting table and onto paper. Manipulatory systems can be devised that

use noncountable coefficients. One such manipulatory system is simply a set of cards bearing digits, which are laid out in a line on the table.

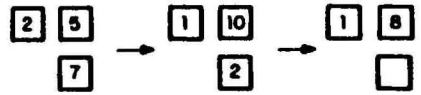
13. *Memorize Pairing Off.* The next few minimal contrasting models are designed to minimize the manipulation of the cards in a manipulatory system, or erasing a digit and writing a new one in a written system. In the previous number systems, column subtraction was realized by pairing off decrements of the top and bottom digits. A "movie" of the card procedure doing 15 − 3 would be



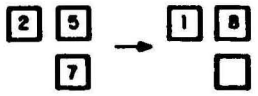
This model replaces this pairing-off loop with a table loopup. A "movie" of the modified card procedure doing 25 − 7 is



14. *Memorize Comparison.* This model procedure replaces the two-step borrowing (see foregoing movie) with a one-step borrow by looking ahead. That is, it looks ahead to see which digit will be zero—the top or the bottom. This amounts to memorizing the greater-than table for digits. Now the movie for 25 − 7 is



15. *Memorize Teens Facts.* Two table lookups can be reduced to one, and two digit rewrites can be saved if a new facts table is provided for the teens facts. The new table is 10 by 9 and contains facts like 15 − 7 = 8. The movie reduces to



16. *Sequence Columns.* In the previous systems, columns are processed in random order. However, this necessitates marking the columns that are done by zeroing the bottom digit. This digit rewrite can be saved if the columns are processed in some set order—either left to right or vice

versa. The planning heuristic—that is, the right difference generator of this mp-morphism—could be called "ordering independent operations reduces marking."

17. *Answer Register.* If a separate place is provided for writing the answer, then erasures of the top digits can be reduced. This is motivated by the fact that writing a digit is easier than erasing—a property peculiar to paper.
18. *Right to Left.* If the columns are processed right to left, one borrows from the top digit. If the columns are processed left to right, one borrows from the answer. The numeral that gets borrowed from ends up with erasures, whereas the other one has no erasures. If one erases by scratching out the digit and writing the new digit above, then the numeral that's borrowed from can become a real mess. The motivation for this analogy is that there is more need for the answer numeral to be legible than the top numeral. Hence, subtraction is more efficient if one processes the columns from right to left.

At last, we have arrived at the standard subtraction algorithm via a sequence of procedures/models where each model in this sequence has an mp-morphism between it and its immediate successor, thus creating a well-structured sequence of analogous models converging to the desired target procedure.

REFERENCES

Allen, F., & Cocke, J. A catalog of optimizating transformations. In R. Rustin (Ed.), *Design and optimization of compilers*. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Brown, J. S., & Burton, R. R. Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 1978, *2*, 155-192.

Card, S. K. *Studies in the psychology of computer text editing systems* (Xerox PARC Rep. No. SSL-78-1). Palo Alto, Calif.: Xerox Palo Alto Research Center, 1978.

Evans, T. G. A program for the solution of geometric-analogy intelligence test questions. In M. Minsky (Ed.), *Semantic information processing*. Cambridge, Mass.: MIT Press, 1968.

Fischer, G., Brown, J. S., & Burton, R. R. *Aspects of a theory of simplification, debugging, and coaching* (BBN Rep. No. 3912). Cambridge, Mass.: Bolt Beranek and Newman Inc., 1978.

Greeno, J., Gelman, G. R., & Riley, M. S. *Young children's counting and understanding of principles*. Paper presented at the meeting of the Psychonomic Society, San Antonio, Texas, November 1978.

Newell, A., & Simon, H. A. *Human problem solving*. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Papert, S. A. *Computer-based microworlds as incubators for powerful ideas* (MIT Logo Lab. Working Paper). Cambridge, Mass.: Massachusetts Institute for Technology, 1978.

Sacerdoti, E. *A structure for plans and behavior*. New York: Elsevier North-Holland, 1977.

Sternberg, R. J. Componential processes in analogical reasoning. *Psychological Review*, 1977, *84*, 353-379.

Tversky, A. Features of similarity. *Psychological Review*, 1977, *84*, 327-353.

Winston, P. H. Learning structural descriptions from examples. In P. H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw-Hill, 1975.

Winston, P. H. Learning by creating and justifying transfer frames. *Artificial Intelligence*, 1978, *10*, 147-173.