

HOW TO STRUCTURE STRUCTURED OBJECTS

Ryszard S. Michalski
Robert E. Stepp

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

ABSTRACT

Past research on automated construction of classifications has been concerned with structuring objects that are characterized solely by attribute-value pairs. The methods stemming from that research did not take into consideration the structural information about the objects nor did they utilize general or problem-specific background knowledge in the process of creating a classification. This paper considers problems of generating classifications of structured objects using knowledge about the inter-relationships of various concepts relevant to describing individual objects as well as object configurations.

1. INTRODUCTION

Creating a classification is typically the first step in understanding and formulating a theory about a collection of observations or phenomena. This process is a form of learning from observation (learning without a teacher) and its goal is to structure given observations into a hierarchy of meaningful categories. Past work on this problem was done mostly under the heading of numerical taxonomy and cluster analysis. It was based on the application of some mathematical measure of similarity between objects, defined over a finite, a priori given set of object attributes. Classes of objects were taken to be collections of objects with high intra-class and low inter-class similarity. The methods assumed that objects are characterized by sequences of attribute-value pairs and that this information is sufficient for creating a classification. The methods did not take into consideration any background knowledge about the relationships among object attributes or global concepts that could be used for characterizing object configurations. Nor did they take into consideration possible purposes of classification.

Since the basis for creating classes was purely mathematical, the resulting classifications were often difficult to interpret conceptually. The problem of interpreting the results was left to the data analyst. In addition, since descriptions of objects are simply attribute-value pairs, the methods were inadequate for creating classifications of structured objects, i.e., objects whose appropriate description involves not only object attributes but also relationships among the object parts as well as attributes of the parts.

The recently developed method of conjunctive conceptual clustering [Michalski and Stepp, 1983] overcomes some of the above problems. Given a set of object descriptions, the method constructs a hierarchy of classes and their descriptions which are in the form of conjunctive concepts. The siblings of any node in the hierarchy are optimized according to a certain global criterion of classification "quality." The method, however, still relies on the attribute-value descriptions of objects. It also has only limited ability to take into consideration the background knowledge of various concepts. This paper addresses the problems of extending the above method in two directions:

- (1) how to form classifications of structured objects,
- (2) how to formulate and utilize background knowledge in creating classifications.

In the following sections, the above problems are characterized by simple examples and a methodology for their solution is outlined.

2. STRUCTURING STRUCTURED OBJECTS

In order to describe this problem in greater detail, let us consider a simple example based on rephrasing the problem known as "East- and Westbound trains" [Larson, 1977; Michalski, 1980a] shown in Figure 1. In the old formulation of the problem, given are two collections of trains, those that are "Eastbound" (A to E) and those that are "Westbound" (F to J). These trains are highly structured, each consisting of a sequence of cars of different shapes and sizes. The individual cars carry a variable number of loads of different shapes. In this formulation, this is a typical problem of *learning from examples* (or *concept acquisition*) in which the task is to automatically formulate a simple rule distinguishing between the two classes of objects.

Suppose now the class labels are removed (as in Figure 1) and the problem is to create a meaningful classification of the collection of trains. This is a form of a *learning from observation* (or *concept formation*) problem. To illustrate a possible solution for the concept formation problem, let us go back to the concept acquisition formulation of the problem and consider discriminant descriptions of Eastbound and Westbound trains found by the program INDUCE/2 [Hoff, Michalski, Stepp, 1983]. These descriptions were:

Eastbound trains:

"A train is Eastbound if it contains a short, closed car."

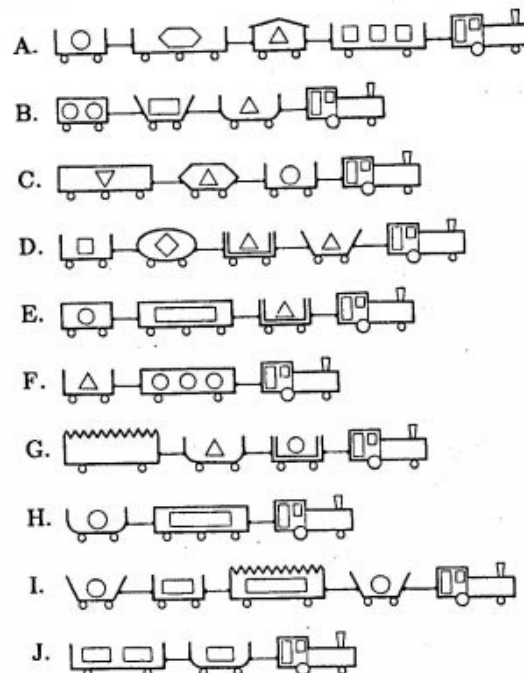


Figure 1. How would you classify these trains?

Westbound trains:

"A train is Westbound if there are just two cars in a train or if a train has a car with a jagged top."

To illustrate the description language used, below is the actual output from the INDUCE/2 program that corresponds to the above descriptions:

Eastbound: $\forall(\text{train}) [\text{contains}(\text{train}, \text{car1})] [\text{length}(\text{car1}) = \text{short}] \& [\text{shape}(\text{car1}) = \text{closed}]$

Westbound: $\forall(\text{train}) [\text{num-cars}(\text{train}) = 2] \vee [\text{contains}(\text{train}, \text{car1})] [\text{shape}(\text{car1}) = \text{jagged top}]$

Various aspects of learning structural descriptions from examples are given in [Winston, 1977] and [Dietterich and Michalski, 1983]. The problem of concern here is to develop a general method that when applied to the collection of trains without class labels (Figure 1) could potentially restore the above classification or invent one of equal conceptual appeal. In addition it should generate descriptions of the same kind as above for the created classes, with a possible proviso that each class is described by a simple conjunction only (in this case we would have three classes of trains: those with a short closed car, those with a jagged-top car, and those with exactly two cars).

Such classification construction problems occur when one wants to organize and classify observations that require structural descriptions. Problems of this type include classifying physical or chemical structures, analyzing genetic sequences, building taxonomies of plants or animals, and characterizing visual scenes. In Sec. 4 we propose two methods for solving this type of problem, one data-driven and one model-driven. Before discussing these methods we turn to the problem of how to use background knowledge in creating a classification.

3. BACKGROUND KNOWLEDGE

It can be observed that when people create a classification of some observations or partition a system into subsystems, they employ knowledge about various concepts and relationships relevant to describing the observations or the system. Let us consider a simple example. Suppose that we are observing a typical restaurant table on which there are such objects as food on a plate, a salad, utensils, salt and pepper, napkins, a vase with flowers, a lamp, etc. Suppose a person is asked to build some meaningful classification of the objects on the table. One way to create a classification is to perform the following chains of inferences:

- salt and pepper are seasonings
seasonings are used to add zest to food
seasoned food is something to be eaten
things which are to be eaten are edible
salt and pepper are edible

- salad is a vegetable
vegetables are food
food is something to be eaten
things which are to be eaten are edible
salad is edible

A similar chain of inferences applied to "meat on a plate" or "cake on a dessert plate" will also lead to the concept "edible." On the other hand, a lamp is not food and is therefore not edible. A vase containing flowers is not food and is therefore not edible. Consequently, one meaningful classification of objects on the table is simply "edible" vs. "inedible." The problem that we pose is this: suppose we are given descriptions of objects on the table in terms of their attributes (including their structural attributes) and we want to have a program that on the basis of these descriptions would create the above classification into edible vs. inedible objects. Obviously, such a program would have to be equipped (among other

things) with the above inference rules and with the ability to use them.

4. PROPOSED SOLUTION

This section describes two methods for solving problems of the kind posed above, one data-driven and one model-driven. The data driven method is based on our previous work and reduces the problem of building a classification into a sequence of concept acquisition problems (specifically, problems of determining discriminant descriptions of objects with given class labels). Thus the method will be called RD for "Repeated Discrimination."

The model-driven method is based on selecting *classifying attributes* (attributes that are used to define classes) either from the initially given pool of attributes or from attributes generated by applying inference rules provided by background knowledge. This method will be called CA for "Classifying Attributes."

The common components of the methods are that they both use general and domain-specific knowledge in the process of constructing a classification and that they both utilize a similar criterion for measuring the "quality" of generated candidate solutions. Therefore, before we launch into a description of the methods, we will first briefly discuss our language for describing generated classes and expressing background knowledge inference rules, and then we will discuss a criterion for measuring the quality of proposed classifications.

4.1. The representation language

In order to build a classification of structured objects, we must have an adequate language for describing such objects as well as their classes. This requirement suggests that we use a predicate calculus or some modification of it. Here we have chosen a language called *annotated predicate calculus (APC)*. APC is an extension of predicate calculus that uses several novel forms and attaches to each predicate, variable, and function an *annotation* [Michalski, 1983]. The annotation is a store of information about the given predicate or atomic function. Together with all forms found in predicate calculus, the language also uses a special kind of predicate called a *selector*. A simple selector is in the form:

[atomic-function REL value-of-atomic-function]

where REL (relation) stands for one of the symbols $= \neq < > \leq \geq$. An example of such a selector is

[weight(box) > 2kg]

which means "the weight of the box is greater than 2 kg." A more complex selector may involve *internal disjunction* (disjunction of values of the same atomic function) or *internal conjunction* (the conjunction of atomic-functions having the same value). These two operators are illustrated by the two corresponding examples:

[color(box) = red $\vee$ purple]	"the color of the box is either red or purple."
[color(box1 & box2) = red]	"the color of box1 and box 2 is red."

Selectors can be combined by standard logical operators to form more complex expressions. For example, a statement "there is a green circle on top of a blue or red square" can be expressed as

$\exists(p_1, p_2) [\text{ontop}(p_1, p_2)] [\text{color}(p_1) = \text{green}] [\text{shape}(p_1) = \text{circle}] \& [\text{color}(p_2) = \text{blue} \vee \text{red}] [\text{shape}(p_2) = \text{square}]$

(Logical multiplication is denoted by the symbol & or by concatenating selectors).

The background knowledge is expressed as a set of APC implicative rules:

CONDITION $\Rightarrow$ CONSEQUENCE

where CONDITION and CONSEQUENCE are conjunctions of selectors. If CONDITION is satisfied, then CONSEQUENCE is asserted. To illustrate the implicative statement, consider the assertion "vegetables are food" from the example in Sec. 3. It can be expressed:

$$[\text{is-vegetable}(\text{object1})] \Rightarrow [\text{is-food}(\text{object1})]$$

An alternative way to express it is

$$[\text{type}(\text{object1}) = \text{vegetable}] \Rightarrow [\text{type}(\text{object1}) = \text{food}]$$

In this expression "vegetable" and "food" are treated as elements of the structured domain of the attribute "type." This implication expresses a generalization inference rule called *climbing the generalization tree* (the domain of the attribute "type" is assumed to have a tree structure). Further details on the APC language are given in [Michalski, 1983].

4.2. Measuring the quality of a classification

Creating a classification is a difficult problem because there are usually many potential solutions with no "correct" or "incorrect" answers. The choice of a classification can be made according to some perceived goal of classification or some measure of the quality of the classification. One way to measure classification quality that has been successful in both INDUCE/2 and CLUSTER/2 is to define various elementary, easy to measure criteria specifying desirable properties of a classification and to connect them together into one general criterion called the *Lexicographical Evaluation Functional* (LEF) [Michalski, 1983]. The LEF consists of an ordered sequence of elementary criteria along with tolerances which control to what extent different solutions are considered equivalent. The elementary criteria measure certain aspects of the generated descriptions such as the fit between the classification and the objects, the simplicity of the descriptions (the reciprocal of the number of selectors), the discrimination index (the number of attributes that singly discriminate between all classes), and the dimensionality reduction (the number of attributes that are necessary to classify the objects).

4.3. How to use background knowledge

Building a meaningful classification relies on finding good classifying attributes (attributes that define classes). The data-driven and model-driven methods described in the next section both use background knowledge rules in the search for such attributes. The background knowledge rules enable the system to perform a chain of inferences that add new attributes to object descriptions during the course of classification construction. These new attributes are tested to determine if they are good classifying attributes.

The background knowledge rules can represent both general knowledge managed by the classification program and available to all problems, and problem-specific knowledge provided each time by the data analyst. In either case, the knowledge is supplied in the form of an inference rule (called a background rule, or *b-rule*). Special types of b-rules include expressions of arithmetic relationships (*a-rules*), such as

$$\text{girth}(\text{object}) = \text{length}(\text{object}) + \text{width}(\text{object})$$

and implicative rules that specify logical relationships (*i-rules*) such as:

$$[\text{above}(p1,p2)][\text{above}(p2,p3)] \Rightarrow [\text{above}(p1,p3)]$$

Each kind is associated with a condition indicating the situations to which the rule is applicable.

Background knowledge is handled somewhat differently in the data-driven and model-driven approaches. This distinction will be characterized below.

4.4 How to form classifications of structured objects

4.4.1. A data-driven approach: concept formation via concept acquisition

This section explains how a problem of concept formation (here, building a classification) can be solved via a sequence of controlled steps of concept acquisition (learning from examples). To begin with, let us briefly describe the program INDUCE/2 which solves concept acquisition tasks involving structured objects.

Given a set of objects described by annotated predicate calculus and arranged into two or more classes, INDUCE/2 generates a description of each class (in the form of annotated predicate calculus expressions) that covers all the objects in the described class and no objects in any other class. This is accomplished in the following manner.

The objects are divided into two sets: objects belonging to the class being described (called set F1), objects belonging to any other class (called set F0). One object at a time is selected from set F1 (the "focus of attention") and a "star" is built that covers the selected object against all objects in set F0. A star is a set of all alternative descriptions that are maximally general, cover the "focus of attention" object (the *seed*) and possibly other objects from F1, and cover no objects from F0 [Michalski, 1983]. To control computational explosion, a parameter MAXSTAR restricts the number of alternative descriptions that are retained during the star generation process. The "best" description in the star (according to a LEF criterion) is selected, and it becomes part of the solution. The objects covered by the selected description are removed from set F1 and the process is repeated by selecting another seed object (from among those not yet covered) and building a star for it. When all objects in the set F1 have been covered, the solution is complete.

The INDUCE/2 algorithm can be adapted for solving classification construction problems. Given a set of unclassified objects, k objects are selected and treated as individual representatives of k imaginary classes. The INDUCE/2 algorithm then generates descriptions of each representative that are maximally general and do not cover any other representative object.

These descriptions are then used to determine the most representative object in each class (defined as the set of objects satisfying the class description). The representative objects are then used as new seeds for the next iteration. This process stops when either consecutive iterations converge to some stable solution, or when a specific number of iterations will not improve a classification (from the viewpoint of the criterion LEF).

The control layer that provides the representatives to the concept acquisition process must be able to pick good representatives or else the resulting classifications will be arbitrary without revealing underlying patterns in the data. One technique that has been used in a variety of clustering situations is to pick representatives at random in the first step. In the next steps *central* representatives or *extreme* representatives are selected alternately [Michalski and Stepp, 1983].

This approach requires the selection of a defined number of representative objects (which corresponds to the number of classes). Since the best number of classes to form is usually unknown, two techniques are used:

- (1) varying the number of classes, and
- (2) composing the classes hierarchically.

Since the classification to be formed should be easy to understand by humans, we assume an upper limit on the number of classes that stem from any node of the classification hierarchy. This limit is assumed to be in the range of 4 to 7. Since this limit is small it is computationally feasible to repeat the whole process for

each number of classes. The solution that optimizes the score on the LEF (with appropriate adjustment for the effects of the number of classes on the score) indicates the best number of classes to form at this level of the hierarchy.

To give an example solution to the "Trains" problem from Figure 1, we show in Figure 2 a classification created by a data-driven approach (using an earlier method described by Stepp [1978]). We find this kind of solution appealing because the difference between classes is striking, yet not obvious by casual inspection.

4.4.2. A model-driven approach

In contrast to the data-driven method described in the preceding section, we will now describe a model-driven approach. The underlying goal of the model-driven approach is to find one or more classifying attributes whose observed value sets can be split into ranges that characterize each class. The important aspect of this approach is that the classifying attribute can be derived through a chain of inferences from the initial attributes during the course of constructing the classification. The classifying attributes sought are the ones that lead to classes of objects maximizing the global criterion of classification quality, (e.g., classes representing simple concepts and/or containing objects that share values for all attributes considered important).

The importance of an attribute can be determined either on the basis of the classification goal or by the fact that it implies many other attributes. For example, if the goal of the classification is "finding food," the attribute "edibility" from Sec. 3 is the important classifying attribute. The second way of determining the importance of an attribute can be illustrated by the problem of classifying birds. The question of whether "color" is a more important classifying attribute than "is-waterbird" is answered in favor of "is-waterbird" because it implies more secondary attributes than does the attribute "color" (e.g., is-waterbird implies "can-swim," "has-webbed-feet," "eats-fish").

There are two fundamental processes that operate alternately to generate the classification. The first process searches for the classifying attribute whose value set can be partitioned to form

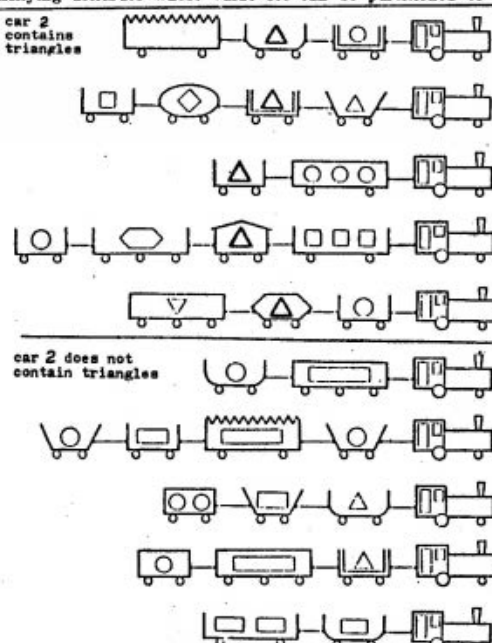


Figure 2. A sample classification of the trains found by a data-driven approach.

classes such that the produced classification scores best according to the LEF. The second process generates new attributes from logical and arithmetic combinations of other attributes, as guided by background knowledge inference rules.

The attribute search can be performed in two ways. When the number of classes to form (k) is known in advance, the process searches for attributes having k or more different values occur in the descriptions of the objects to be classified. These values are called the *observed values* of the attribute. Attributes with the number of observed values smaller than k are not considered.

For attributes with observed value sets larger than k , the choice of the mapping of value subsets to classes depends on the resulting LEF score for the classification produced and the type of the value set.

For attributes with nominal (unordered) value sets, any combinations of values can be mapped to classes. For attributes with linearly ordered value sets, only non-overlapping closed intervals of values are mapped to classes. For attributes with generalization-tree ordered value sets [Michalski, 1983], sibling values in the generalization tree are mapped to classes.

For example, if k is 4 and the two attributes x_1 and x_2 can have 7 and 3 values, respectively, only x_1 is a candidate for determining the classification. The selection of which of the 7 values denote which of the 4 classes depends on the type of the value set. If we assume that the value set is linearly ordered, one possible classification involving 4 classes could be

$x_1 = 0..1$	(class 1)
$x_1 = 2..3$	(class 2)
$x_1 = 4$	(class 3)
$x_1 = 5..6$	(class 4)

This classification would be selected if it is best according to the quality criterion LEF (which also takes into consideration values of other attributes).

When the number of classes to form is not known, the above technique is performed for a range of values of k . The best number of classes is indicated by the classification that is best according to the LEF.

The attribute generation process constructs new attributes from combinations of existing attributes. Certain heuristics of attribute construction are used to guide the process. For example, two attributes that have linearly ordered value sets can be combined using arithmetic operators. When the attributes have numerical values (as opposed to symbolic values such as "small," "medium," and "large") a trend analysis can be used to suggest appropriate arithmetic operators as in BACON 4 [Langley, Bradshaw, Simon, 1983]. Predicates can be combined by logical operators to form new attributes through l-rules. For example, the rule

$[\text{cold-blooded}(a1)][\text{offspring birth}(a1)=\text{egg}] \Rightarrow [\text{type}(a1)=\text{reptile}]$

yields a new attribute "type" with a specified value "reptile." Using this rule and similar ones, one might classify some animals into reptiles, mammals, and birds (even though the type of each animal is not stated in the original data).

5. SUMMARY

We have discussed problems of creating a classification of structured objects and have outlined a data-driven method and a model-driven method. The first method transforms concept formation into a sequence of concept acquisition tasks. The second method forms classes by partitioning the value set of one or more classifying attributes that were determined as most appropriate according to a classification quality criterion (LEF). The classifying attributes are either selected from the initially given ones or derived through an application of inference rules defined in the problem background knowledge. Both approaches make extensive use of

attribute constructions through inference rules provided by general and problem-specific background knowledge. The capability to incorporate background knowledge in classifying structured objects adds a new dimension to the tasks of concept formation and data analysis.

REFERENCES

- Dietterich, T.G. and Michalski, R.S., "A Comparative Review of Selected Methods for Learning from Examples," chapter in *Machine Learning*, R.S. Michalski, J. Carbonell and T. Mitchell (Editors), Tioga Publishing Company, 1983.
- Hoff, W.A., Michalski, R.S., Stepp, R.E., "INDUCE 2: A Program for Learning Structural Descriptions from Examples," Technical Report, Department of Computer Science, University of Illinois, Urbana, Illinois, September, 1983.
- Larson, J., "Inductive Inference in the Variable-Valued Predicate Logic System VL₂₁: Methodology and Computer Implementation," Ph.D. Thesis, Report No. 869, Department of Computer Science, University of Illinois, Urbana, Illinois, 1977.
- Langley, P., Bradshaw, G.L., Simon, H.A., "Rediscovering Chemistry With the BACON System," chapter in *Machine Learning*, R.S. Michalski, J. Carbonell and T. Mitchell (Editors), Tioga Publishing Company, 1983.
- Michalski, R.S., "Pattern Recognition as Rule-Guided Inductive Inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, NO. 4., pp 349-361, July, 1980a.
- Michalski, R.S., "Knowledge Acquisition Through Conceptual Clustering: A Theoretical Framework and an Algorithm for Partitioning Data into Conjunctive Concepts," *Policy Analysis and Information Systems*, Vol. 4, No. 3, pp. 219-244, 1980b.
- Michalski, R.S. and Stepp, R.E., "Concept-based Clustering versus Numerical Taxonomy," Technical Report 1073, Department of Computer Science, University of Illinois, 1981.
- Michalski, R.S., "A Theory and Methodology of Inductive Learning," chapter in *Machine Learning*, R.S. Michalski, J. Carbonell and T. Mitchell (Editors), Tioga Publishing Company, 1983.
- Michalski, R.S. and Stepp, R.E., "Learning From Observation: Conceptual Clustering," chapter in *Machine Learning*, R.S. Michalski, J. Carbonell and T. Mitchell (Editors), Tioga Publishing Company, 1983.
- Winston, P.H., *Artificial Intelligence*, Addison-Wesley, 1977.

THE ARCHITECTURE OF JUMBO

Douglas R. Hofstadter
Computer Science Dept.
Indiana University
Bloomington, Indiana 47405

ABSTRACT

Jumbo is an artificial-intelligence project concerned with the reshuffling of parts to form different coherent or "well-chunked" wholes (specifically anagrams, but without using a dictionary). It is based on the concept of a parallel nondeterministic terraced scan: a somewhat Hearsay-like method of homing in, via cues at many levels, to the best category to which to assign something. The process is guided by the "happiness" of structures it creates. A structure's happiness is a measure of how easy it is to accommodate the Platonic concept it is currently considered to be an instance of, to it. For Jumbo to discover good solutions, its data structures must be fluidly reconfigurable, and this is achieved by regrouping, rearrangement, and disbanding. The overall behavior of Jumbo is not directly programmed, but is an emergent, or statistical, outcome, of the interaction of the microscopic computational activities happening in parallel.

Key terms: Perception, recognition, parallelism, artificial intelligence, terraced scan, fluid data structures, temperature, self-watching, emergence.

PROLOGUE: NOT MUMBO-JUMBO

Jumbo is an artificial-intelligence research project with a concrete purpose and an abstract motivation. It is one of several projects, all focused on various facets of one shared long-term goal [Hofstadter 1983]. In essence, that goal is to demonstrate the equivalence of cognition with "deep perception" -- the non-modality-specific layers of perception that involve funneling into highly abstract and often unverbalizable categories [Schank 1980, Hofstadter 1979, 1981b, and 1982b]. The slogan "cognition equals recognition" captures the idea.

Deep perception (recognition) [Bongard 1970, Hofstadter 1982c] is an unsolved -- in fact, almost untouched -- problem of artificial intelligence. It is a far cry from low-level vision; it is the murky area where syntactic sensations interface with semantic categories. This syntactic-semantic transition zone, I feel, is the core mystery of all of intelligence. Before we can make machines that can learn, we must understand how concepts

are structured and can be addressed and compared. Therefore Jumbo is a prelude to a more ambitious project in learning.

JUMBO AND THE ANALOGIES IT IS BASED ON

Jumbo's concrete purpose is to imitate human skills used in the anagram game "Jumble" [Arnold 1982]. In this game, a human player seeks an English word made out of a few given letters. The idea of the Jumbo system is for a computer to construct English-like word candidates ("pseudo-words") from a given set of letters. Jumbo possesses no dictionary of English; it makes its constructions purely by referring to stored knowledge defining how consonant and vowel clusters are formed out of letters, syllables out of clusters, and words out of syllables. Thus Jumbo is a building program; it begins with isolated atomic units (letters), and gradually constructs conceptual molecules or chunks ("gloms") out of them. There are molecules at various levels: clusters, syllables, and words.

Why no dictionary? Because that is irrelevant to the mental processes I am attempting to model, namely the unconscious composition of coherent wholes out of scattered parts. A dictionary could be added, but it would be irrelevant to Jumbo's serious purpose, which is to be a model of the mental process of assembly and transformation -- not an expert system for playing a word game. The hope is that these processes (and an architecture allowing their smooth integration) are notions exportable to more complex domains.

The strategy of Jumbo is based on two analogies. One is to the way that complex molecules are constructed inside a living cell [Lehninger 1975]; the other is to the way that bonds of human friendship are formed in a chaotic world.

I will describe the biology analogy first. In a cell, the lowest level of molecule is the atom. Above that, there are very small molecules, which might be compared to very tight consonant clusters (such as "th" or "ng"). The next level up is that of, for instance, amino acids. These could be compared to loosely bound clusters, such as "str" or "ngth" or "oo". Then there are linear chains of

amino acids -- polypeptide chains (often complete proteins). They could be likened to syllables (often complete words). Many proteins (words) consist of an agglomeration of several polypeptide chains (syllables).

The assembly of molecules in a cell takes place throughout its cytoplasm (the major part of the interior of the cell). For each molecule type, there is a standard chemical pathway by which it is assembled. The assembly pathway for a given molecule type may involve dozens of steps that build it piecemeal. All over the cytoplasm, that pathway is followed by separate, independent, parallel but non-synchronized assembly processes.

Next I will briefly sketch out the "friendship" analogy -- or perhaps the "romance" analogy would be a better term, since the building of romances is the main driving image. In society, the atomic units are individual persons. Then there are two-person units (couples). Above that there are friendships among couples, and larger societies. This metaphor, however, has less to do with the multi-leveledness of society than with the time-dependent manner of formation of bonds. The basic idea is that it takes experimentation ("flirtation" and "dalliance") for two people to decide if they wish to be romantically involved.

The first stage is chance-dependent: do the two individuals come near enough to notice each other? If not, there is no hope of a bond forming. If so, this initial noticing may generate the germ of a romance: a "spark" propelling the partners into a desire for further exploration. If they survive this next stage of flirtation with each other, then I call what results a "flash": a mutual infatuation not yet resulting in a bond. The next stage is that of casual dating ("dalliance"), determining compatibility in a more serious way. If they survive this stage, they can form a genuine bond -- a "romance". There is still a further stage beyond this, of course, called "marriage".

Romantic relationships often break up. Some breakups result from internal tensions in the relationship; others are instigated by the arrival of a seemingly more desirable partner. If an attractive potential partner turns up, many things have to be seriously weighed against each other. The outcome depends on the internal "happiness" of the marriage, the excitement promised by the potential partner, and the mood of society regarding divorce.

In the cell, too, breakups of molecules occur constantly. Large molecules break into smaller constituents, and from these pieces new molecules are then assembled. Assembly and disassembly are not spontaneous but are mediated by agents that are complex molecules in themselves -- enzymes. For each type of operation, there is a specific type of enzyme.

THE SIGNIFICANCE OF JUMBO'S TASK DOMAIN

Now what is significant about coming up with some English-like rearrangements of a set of letters? Well, the perceptual process is essentially one of constructing larger units out of smaller ones, with temporary structures at various levels and permanent mental categories trying to accommodate to each other. A lot of back-and-forth motion must occur -- that is, construction, destruction, regrouping, and rearrangement of structures. The architecture for a system to carry out this process is the result of many subtle decisions about how independent processes should interact, how structures should be put together or broken apart, what kinds of things form stable structures, what easy ways are of making new possible structures when old ones are seen to be inadequate, and so on [Lea 1980, Hanson 1978, Waterman 1978].

The Hearsay-II speech understanding system [Reddy 1976, Erman 1980] was a sophisticated example of such an architecture; Jumbo is an attempt to construct another such architecture, and to focus clearly on the essence of the matter. That is one reason why the domain is kept small.

JUMBO AND PARALLELISM

Jumbo is a parallel system, yet its parallelism, based on the cell, diverges from that of some models of parallelism. In a cell, many metabolic activities take place simultaneously at different spatial locations. Each activity, whether anabolic (i.e., bond-forming) or catabolic (i.e., bond-breaking), is carried out by an enzyme. The typical anabolic enzyme's action involves the bonding together of two molecules. How does the enzyme find those molecules? It has "crevices" called "active sites" that match only molecules of the right type. It wanders around the cytoplasm, and when it bumps into a structure that fits one of its active sites, it latches onto it, and then the two wander together until the enzyme's second active site gets similarly filled at random. At that point, the enzyme is triggered into action. It performs its joining operation, and releases its product. This product is then potential material for a new enzyme to graft something else onto it. Thus larger pieces get built from smaller ones, in a random way. Yet despite the randomness, specific building tasks defined by elaborate chemical pathways get carried out efficiently and reliably.

Jumbo's parallelism is modeled on the distributed parallelism of the cell. A salient difference is that Jumbo runs on a sequential computer, so its parallelism must be virtual rather than actual. However, Jumbo could be adapted to run on a distributed parallel computing architecture of the right sort.

At the outset, all the letters are "lonely" and clamoring for attention. What chooses who gets to

flirt or bond with whom? In essence, it is random, as in the cell or the real social world. To visualize Jumbo, imagine the letters floating around in a three-dimensional fluid -- the "cytoplasm". If two letters come close to each other, they can "see" each other and possibly "spark".

What determines whether a spark is successful? At the outset, when all letters are unattached, it is affinity ("chemistry", as the intriguingly accurate popular metaphor would have it). For example, "s" and "h" (in that order) are well suited, but in the opposite order, they are not. Affinity is not all or none. Some letters are very happy with each other ("s" and "h"), some only mildly happy with each other ("d" and "w", as in "dwell"), and some not at all happy together ("j" and "x"). There are all degrees of affinity. Correspondingly, some sparks are strong, some weak, and some fizzle.

What is the computational meaning of "sparking"? A spark is a data structure telling who is sparking with whom, and in what order. Manufactured at the same time as the spark and associated with it is a piece of code called a "codelet", which is placed in a queue-like structure called the "coderack", so that it can be run. The codelet, when run, will look at the spark, evaluate its viability, and then suggest whether it is worthwhile going on with further exploration in this tentative "romance". If this flirtation fails, then both letters will go on their merry ways, each one free to spark with other partners instead. If the flirtation seems promising, though, then the codelet will create a "flash" -- the next stage of a romance.

Which codelet on the coderack shall run first? Associated with each codelet is a number, its "urgency". The choice of which codelet to run next is a weighted but random choice, the urgency of a codelet being its weight in the coderack. Thus a codelet with urgency 10 is five times as likely to be picked to run as a codelet with urgency 2. But there is no guarantee that it will be run first. It is all probabilistic, so that only in a statistical sense will what "should" happen more often actually happen more often.

Whenever a codelet is run, it is removed from the coderack. The only traces it leaves behind are: (1) changes that it has caused in the cytoplasm; and (2) new codelets it has placed on the coderack. This enables lengthy processes to be carried out in small disjoint steps, each step setting up its own possible continuation. It resembles the way that long chains of chemical reactions get carried out in independent small steps in the cell.

THE CONCEPT OF A TERRACED SCAN

Why a spark, then a flash, then perhaps a dalliance, and so on? Why not check out the whole thing at once? The answer is: real-time pressures.

If this potential union were the only possibility in the world, then time wouldn't press. But if there are many possible options to choose from, time does indeed press. One needs to circumvent what at first might seem an awful necessity: that of exploring all potential matchups to an equal degree before making up one's mind about which one works best. There is a profound need for protecting oneself from this sort of absurdity. People develop ways of quickly eliminating things of little interest to them and homing in on the good possibilities. This is the idea of the "terraced scan": a parallel investigation of many possibilities to different levels of depth, quickly throwing out bad ones and homing in rapidly and accurately on good ones.

The terraced scan moves by stages: first one performs quick superficial tests, and one proceeds further only if those tests are passed. Each new stage involves more elaborate tests, which, if passed, can lead to a further stage -- and so on. Furthermore, "passing a test" is not an all-or-nothing affair; each test produces a score, indicating how promising this line of investigation appears so far. One uses this score to determine the urgency of continuation codelets. This provides the desired layeredness to the evaluation of the goodness of a potential matchup.

Since Jumbo is serial, it must interleave the tests making up the parallel explorations, so that many possibilities can be simultaneously explored. Each stage determines the urgency of its follow-up stage, so that the speed of a long process is locally adjustable to reflect the promise it offers.

The way Jumbo works, then, is that it examines many possibilities in simulated parallel, letting the more promising ones move ahead faster than the less promising ones. Letters begin by sparking with one another. Each successful spark engenders a flash. Each successful flash engenders a codelet which -- if and when run -- will bond the given letters together into a new structure -- a glom. Gloms, like letters, also spark together, flash together, and can be bonded together into larger and larger gloms.

While two given items are sparking, two others may be flashing. Others may already have been bonded together to make a glom. Although any particular "romance" has to go through a fixed sequence of increasingly elaborate testing stages, many such explorations may be happening in parallel, and need not be in phase. Flash-evaluators, spark-evaluators, and other types of codelets commingle and coexist in the coderack without conflict.

Furthermore, nothing prevents a given item from sparking with several other items at once. Thus parallel exploration of more than one "mate" can be carried out. Up to a point, a given item can pursue two romances at once, just as a person can pursue two

romantic possibilities at once, by interleaving the dates and letting the rival romances develop in parallel. There comes a point, however, where if things are to go any further in a romance, commitment is necessary. This means bonding.

GLOMS, OR COMPOUND STRUCTURES

The idea of bonding is to create compound structures that have stability and that are difficult to penetrate or break apart. Typical gloms (bonded structures) include consonant and vowel clusters, syllables, and words. All of these may be incomplete or imperfect. Examples of incomplete or imperfect structures are "ngt", an incomplete consonant cluster (it needs an "h" at the end), and "oot", "y", and "ies", which are incomplete syllables (they need opening consonant clusters).

A bond-making codelet is like an enzyme in a cell causing the bonding of two pieces into a larger piece. The new piece contains the two previous pieces as parts. The parts are still accessible to the outside world, but not as easily as before. Moreover, the new structure now can enter into new kinds of relationships that neither part was entitled to.

Any new glom in the cytoplasm is given a name. Accessible through that name are facts such as the names of the constituents of the glom, the type of the glom (is it a syllable, word, etc.), the "happiness" of the glom (is it complete or not? how happy are its parts?), and so on. To the lowest approximation, being encapsulated inside a glom renders an item inaccessible to the outside world, much as being married -- at least in theory -- prevents a person from having new romances. In that sense, a new glom effectively replaces its pieces in the cytoplasm.

However, the creation of a new glom does not set off a massive search in which all codelets referring to parts of the new glom are purged. Those codelets are allowed to stick around, and when run they will usually just fizzle harmlessly. It is like learning that someone you find attractive is involved; usually you just give up with a sigh.

It is possible, though, that a piece inside a stable glom sparks so strongly with an external piece that the result challenges the bond holding the first piece in. Depending on the "happiness" of the glom, the challenge may be defeated or it may emerge victorious, which will result in the dissolution of the bonds making the old glom, and the establishment of a new glom. Thus in Jumbo, being "committed" (bonded) acts as a damper, but not a total inhibitor, of exploration of rival romances -- as is the case with people.

PARALLEL CONTEMPLATIONS VERSUS PARALLEL REALITIES

Jumbo aims at converting the contents of the cytoplasm from an initial array of unconnected and unrelated letters into a single final structure -- a coherent and hierarchically structured assembly of parts, a candidate for "wordhood". Hearsay II also built hierarchical structures out of an initially formless chaos, but with one salient difference: it constructed multiple top-level structures -- rival interpretations of what the speaker was saying -- at once. An analogous vision program would construct several top-level interpretations for a scene in parallel.

This strikes me as implausible. We do not experience a cognitive superposition of states, seeing (or hearing) something two or more ways at once. The famous examples of visual ambiguity -- the flipping Necker cube, the vase-face picture, and many others -- substantiate my point. While over time we can easily flip back and forth between top-level points of view, our cognitive systems do not easily support seeing something in two ways at one moment.

This suggests that parallelism becomes less and less relevant at progressively higher levels of activity. This idea is mirrored in the way Jumbo works. As structures coalesce, the system gradually makes a transition from being distributed and parallel to being localized and serial; at the end, there is only one top-level glom left in the cytoplasm. Two gloms are never just different arrangements of the same lower-level structures. Therefore, a certain kind of parallelism is nonexistent in Jumbo -- namely, the parallelism you get when you entertain alternative worlds. Jumbo has but one world (its cytoplasm). However, parallel contemplation of alternative worlds (i.e., possible actions to try in the future) is permissible in Jumbo; in fact, it is of the essence. Jumbo is free to "muse" about many possible rival actions, but once one is chosen and carried out, then that's the way the world is.

Let us thus draw a distinction between two types of codelet. A "musing" codelet contemplates a future possibility without actually creating it; an "action" codelet realizes a contemplated possibility by carrying out a change in the cytoplasm. Musing codelets perform "read-only" operations on the cytoplasm (e.g., evaluating sparks or flashes); action codelets perform "write"s in the cytoplasm (e.g., making, breaking, strengthening, or weakening a bond).

In summary, in the early stages of glom-building, when Jumbo's cytoplasm is composed of many scattered pieces, nonoverlapping actions in it can be viewed as occurring simultaneously (whether the simultaneity is virtual or real); however, as gloms merge and become larger but fewer in number,

parallelism of action inside the cytoplasm gradually gives way to seriality, although the parallel contemplation of possible actions to take continues unabated.

THE HAPPINESS OF GLOMS

At any stage, Jumbo's cytoplasm contains gloms having various degrees of "happiness" -- a numerical measure applicable to all gloms. There are two components to happiness. The INTERNAL component tries to summarize how well the glom, as a specific instance of a general concept, realizes that Platonic ideal. The EXTERNAL component tries to summarize the current situation of this glom in the cytoplasm. Is it part of a larger glom (thus "secure"), or is it unattached (thus "lonely")?

Consider a syllable-level glom. It must ask itself, "How good a syllable am I?" This question is answered by a number taking into account the degree of completeness of the glom, the way its pieces fill the roles they are supposed to play, the suitedness of its parts for each other, and -- recursively -- the internal happinesses of its parts.

Take the syllable-candidate "lik", for instance. This glom, seen as a potential syllable, is complete: it has a filler for the three main roles required for "syllablehood": initial consonant cluster, vowel group, and final consonant cluster. (Such roles are analogous to the roles of "husband" and "wife" in a marriage, although not all three of them have to be present in every syllable, "ing" being a good example of an internally happy syllable lacking an initial consonant cluster.) Being a COMPLETE syllable is not enough to make "lik" an internally HAPPY syllable, because "k" does not finely fill the "final consonant cluster" role. By contrast, "lick" is an internally happy syllable, since "ck" is a fine final consonant cluster.

Internal unhappiness makes a glom "squeak". For instance, the incomplete syllable "li" wishes to be made complete, so it squeaks quite loudly. The imperfect syllable "lik" will squeak, but not quite so loudly.

In real life, George may ask himself, "How good a husband would I make?" without having any particular wife in mind. He may feel that he fills the abstract "husband" role ideally without there being any specific marriage into which he fits. An analogous notion holds for gloms. Take the syllable "lick". Internally, it is great, but if it has not been incorporated into a larger structure, it will still be unhappy. This brings us to the external component of happiness.

The external component of any glom's happiness depends on how the glom fits into the current state of the cytoplasm. Initially, all letters are unhappy but not desperate, because they are all in the same

boat -- unmatedness. As some get mated and others don't, however, a new "emotion" enters the picture: jealousy. The longer a letter goes unmated, the unhappier it becomes. It "squeaks" to express its unhappiness. This goes as well for unmated higher-level gloms.

"Squeaking" translates into likelihood of sparking with other items in the cytoplasm -- that is, the urgency of a codelet that will place the item on a queue of potential "sparked". As in real life, it's the squeakiest wheel that gets the oil.

"Happiness" is thus a measure that a glom computes, taking into account both its self-estimated internal stability, and its self-perceived "normalcy" relative to the external society. Jumbo's overall goal could be succinctly stated as: producing one happy top-level glom. En route to this goal, it has to be guided by the happinesses of the intermediary structures it makes.

DESTRUCTIVE OPERATIONS IN JUMBO

It would be nice if the randomly-driven building processes that take place from the ground up always gave rise to an excellent candidate for wordhood. However, since this is unlikely, there must be ways of changing around what has been constructed. Since much care has gone into the buildup of the constituents on various lower levels, it would be reckless to disband the thing without trying some reworking first.

Traditionally in artificial intelligence, reaching a decision point where one sees that one has to go back and undo something is a critical moment. The standard solution is "intelligent backtracking" [Bobrow 1974], in which you either take a different fork at that decision point or backtrack further. The latter amounts to undoing one decision after another until you find one you can live with, and then starting there from scratch.

Taking a different fork requires you to choose at random or by reason. The latter is obviously preferable, but the world does not always give you enough time or knowledge to reason everything out. Even the best-calculated time of the fork may lead to no success, in which case you would have been better off choosing at random, for you wouldn't have invested so much time attempting to foresee the unforeseeable. Thus the advantage of choosing at random is that you don't worry over matters that you can't possibly anticipate -- you simply plunge forward and see what happens.

This is what Jumbo does. However, this should not be confused with blindness. Although the "choices" at branch points are made at random, there are powerful biases built in. Not everything has an equal chance; indeed, smart ideas will tend to be assigned high urgencies, and dumb ones low urgencies.

There is no guarantee of this, nor is there be a guarantee that high-urgency codelets will get run first, since after all, randomness is randomness -- but in the long haul, things will work out favorably. This, at any rate, is the hope on which Jumbo is based.

There is another aspect to the hope placed in randomness: it seems more likely that interesting routes will be discovered by accidental exploration than if the "best" route is always chosen at every junction. Furthermore, having an extremely simple control structure allows the system to do things that a very complex control structure would get thoroughly tangled up in. Complex control structures are highly brittle, whereas probabilistic control structures are rubbery and don't care what happens in what order.

So, what are the possible remedies to choose from, when there is an unhappy pseudo-word? There are two basic types of transformation: "entropy-preserving" and "entropy-increasing". By "entropy", I mean "perceived disorder". A cytoplasm containing no bonded letters has maximal entropy; a cytoplasm containing just one pseudo-word has minimal entropy. Entropy-preserving transformations do not increase disorder: they preserve the number of gloms. By contrast, entropy-increasing transformations destroy gloms, bringing the cytoplasm closer to its initial state of total disorder.

Of the two varieties of entropy-preserving transformation, the milder one is REGROUPING: readjusting internal boundaries so as to yield new substructures without rearranging anything. The other, more drastic, variety is REARRANGEMENT: taking pieces of a glom and shuffling them around to make new gloms on the same level [Holland 1981]. A more radical remedy is the entropy-increasing transformation of DISBANDING: pulling a glom's top-level bonds apart and perhaps even disbanding some of the lower-level structures.

ENTROPY-PRESERVING TRANSFORMATIONS

Suppose that the unhappy pseudo-word is "pangloss", perceived as two syllables -- "pang" and "loss". First consider regrouping. One possibility is to move the "g" from the first syllable to the second, thus making "pan-gloss". On the other hand, it is not possible to detach the entire consonant cluster "ng" from the first syllable and to reattach it to the second consonant cluster, because "ngl" is not an acceptable consonant cluster.

A more complex type of regrouping can occur when two syllables coalesce into one, or one breaks into two, like drops of liquid. A simple example can be given using four letters found inside "pangloss". Suppose the two syllables "ap" and "so" get bonded together in this order: "so-ap". Now "soap" might be bisyllabic (like "react" or "Moab"), but its preferred perception is as a single syllable.

Therefore Jumbo should be able to carry out this type of regrouping fluently. The "fission" process -- moving from one-syllable words like "soap" to two-syllable words "so-op" or "so-ap" -- should be as natural for Jumbo as the "fusion" process.

The next entropy-preserving alternative, rearrangement, offers many possibilities. One would be to perform the operation known as "spoonerism": interchanging the initial consonant clusters of two syllables. This produces a new two-syllable candidate, "lang-poss". The term comes from the name of an Anglican minister, The Reverend Spooner, but it is tempting to ignore that dignified etymology and to invent parallel culinary terms: "forkerism" (the interchange of final consonant clusters) and "kniferism" (the interchange of vowel groups). Operating on "pang-loss", those would produce "pass-long" and "pong-lass" respectively. Spoonerism need not involve entire clusters; sometimes pieces of clusters can be exchanged, as in the transformation of "slag-pons" to "lag-spons".

Rearrangement can involve the EXCHANGE of syllables ("pang-loss" to "loss-pang"). It can also involve the internal REVERSAL of a syllable at the cluster level ("stan" to "nast") or the letter level ("stan" to "nats"). Reversal at either level, unfortunately, poses tricky problems. Obviously, not all syllables can be reversed at the cluster level, since some consonant clusters cannot function initially, while others cannot function finally (try "knots", for example). Reversal at the letter level is equally knotty (try "thump"). Thus one cannot just barge ahead with either type of reversal; musing codelets must prepare the way by checking out the desirability of contemplated actions. This emphasizes once again that, although Jumbo is permeated by randomness, it is also permeated by dynamic tests imbuing it with caution and foresight.

Another type of rearrangement is where one syllable wants to trade its final consonant cluster for another one's initial consonant cluster. For instance, in "plag-noss", "plag" might want to trade its "g" for "noss"'s "n", to yield "plan-goss". Again, the feasibility of such a maneuver depends on whether the given clusters can function in the desired new positions. A natural nomenclature for such trades suggests itself: "sporkerism" and "foonerism".

A musing codelet proposing a trade resembles an offer by one baseball team to trade its pitcher for another team's left-fielder. Such offers are usually made out of recognition of weaknesses, and the feeling that both can benefit from the switch. The weaker a team is in a certain position, the more disposed it is to give away something in order to fill that role well. Likewise in Jumbo. The more valuable a trade is perceived to be by the gloms involved, the more urgent will be the corresponding action codelet manufactured by the musing codelet.

Thus the principle of the terraced scan is at work at this stage of the game no less than in the earlier, glomming-together stages.

FLUID DATA STRUCTURES: A PRINCIPAL AIM OF JUMBO

The combination of regrouping and rearrangement provides a surprisingly powerful tool. It preserves a great deal of structure at all times, and yet achieves a significant reshuffling of high-level pieces. Here is a random walk through pseudo-word space, beginning at "pang-loss":

pang-loss	start
pong-loss	kniferism
long-pass	spoonerism
pass-long	switch syllables
pas-slong	regroup "s"
sap-slong	reverse syllable
slap-song	spoonerism
slang-sop	forkerism
sop-slang	switch syllables
slop-sang	spoonerism
slos-pang	trade "p" for "s"
los-spang	spoonerism
loss-pang	regroup "s"
pang-loss	switch syllables
pan-gloss	regroup "g"

You can continue with this process for a long time.

One might suppose this process to be idiosyncratic to the domain, an obscure type of transformation having little relevance to general intelligence. Such an impression would, however, be quite wrong. Recall that Jumbo's deep aim is to aid in demonstrating the thesis "cognition equals recognition". Now any study of recognition reveals the indescribably fluid boundaries of percepts and concepts. There are thousands of ways of drawing the letter "A" (see HOF-6), yet we recognize any of them at a glance. For any mental category (concrete ones like "dog" and abstract ones like "metaphor"), we immediately recognize thousands of instances, despite vast differences.

To recognize -- i.e., to match an item with a Platonic abstraction -- there must be room for flexing both the Platonic abstraction and the mental representation of the thing itself, to make them meet. What kinds of distortion are sensible? Readjustment of internal boundaries, reassessment of connection strengths, and rearrangement of parts are at the core of the recognition process. Thought's fluidity emanates from the nonrigidity -- the fluidity -- of representational structures. They have an ability to adjust, to alter themselves effortlessly, to assume many conformations. For fluidity to emerge at a high level, the underlying data structures must be permeated by an ability to slip back and forth at the drop of a hat.

It is important that this capacity reside inherently inside the structure itself: it must be ready and willing to be internally reformed if you just "tap" it the right way. The term "change of representation", used in some AI projects, usually refers to a "reasoning engine" converting one representation into another representation. With Jumbo, by contrast, reformatibility is inherent in the structures themselves. Perhaps the major part of the intelligence of the system as a whole resides in the hinge-points of its data structures. The import of that key idea goes far beyond the confines of this one domain.

It is my belief that a nearly constant background activity of creative minds is a playful twisting-around of mental structures of all sorts -- turning them inside out or backwards, regrouping parts, erasing or inserting new levels of structure, moving things back and forth between levels, and so on. In short, taking full advantage of the fluidity of their mental representations, and simply being sensitive to unanticipated side effects.

ENTROPY-INCREASING TRANSFORMATIONS

If Jumbo deems its single top-level glom unsatisfactory, its first attempted remedy will be to create regrouping and rearrangement codelets and to give them high urgencies. Thus, a bit of shuffling-about as illustrated above will take place. During this, Jumbo will monitor the overall happiness of the cytoplasm. If after a while that happiness has not reached a satisfactory level, Jumbo will alter its priorities and pour disbanding-codelets into the coderack.

An example of disbanding would be: smash the word candidate into its syllables, then smash those syllables further into their cluster-level constituents. In the case of "pang-loss", this yields: "p", "a", "ng", "l", "o", and "ss". Now one may hope that some fresh gloms will bubble up. If not, one can disband further!

Action codelets for disbanding are called "dissolvers". A disbanding operation is not undertaken lightly, since it risks destroying a happy glom. Therefore, a musing codelet must check the idea out first, and if it approves of the disbanding, it creates a dissolver -- but if there is hope that some gentler remedy might make the cytoplasm happier, it will assign the dissolver a low urgency.

There are two sorts of dissolvers: targeted and nontargeted. The former is aimed at a specific glom that has given trouble for a while. The latter is a beast whose mission is to find ANY glom and tear it into its immediate constituents.

Why nontargeted dissolvers? Well, if you have been troubled by a glom that has been unfruitful no matter how you shuffled it about, one way to restart

quickly would be to load up the coderack with a bunch of nontargeted dissolvers. In a flash, they, not unlike a horde of hungry piranhas, will tear that pesky glom from limb to limb, leaving you to start out again with a totally disordered cytoplasm.

There is a problem here. Suppose you put ten nontargeted dissolvers in the coderack, and after just five have run, the cytoplasm has been reduced to raw letters. Five nontargeted dissolvers remain in the coderack. What's to prevent them from later being called upon to run, and at that time totally ruining the new emerging order?

TEMPERATURE AND SELF-WATCHING

One way is to have a global variable called the "temperature" of the cytoplasm, low when the top-level gloms are happy, and high when they are unhappy. The happiest state is when there is just one single happy glom. Since we want to preserve such a state, this state is assigned the "freezing" temperature. When the cytoplasm is freezing, dissolving is prevented.

When the cytoplasm is boiling, nontargeted dissolvers have permission to tear apart any glom, whereas when it is merely warm, they are inhibited (i.e., have a low probability of carrying out their mission) in proportion to the happiness of the gloms they are attacking. The cytoplasmic temperature is thus Jumbo's equivalent to the "mood" of society regarding divorce. High temperature encourages split-ups, low temperature discourages them.

Cytoplasmic temperature is a very general concept. It provides quick global feedback about the system, helping to dynamically govern the probabilities of various kinds of activity. The temperature regulates the degree of chaos in the system, something like the rods in a nuclear reactor. When things start to get too wild, you lower the temperature a bit, and when things seem to be going nowhere, you raise it and pump in some craziness [Smolensky 1983].

Temperature will be useful only if kept updated. If not assessed frequently enough, its outdatedness could be harmful. Now temperature-measurement codelets must vie for recognition in the coderack along with everybody else. This brings up the tricky question of how the system keeps itself balanced, with so many complex things happening at once, all mingling in the same coderack.

How can loop-like behavior be obviated? One way is for the system to watch its own cytoplasm and coderack [Hofstadter 1982a], and to make sure that if any loopiness is detected, preventive actions are taken. Such simple remedies as raising or lowering the temperature may suffice, as would changing the values of other important global variables.

But how does self-monitoring really take place? This is as yet unexplored territory in Jumbo. The general idea is that there should be a way of summarizing the contents of the cytoplasm and coderack in simple terms, so that no detailed self-scrutiny is needed. For the coderack, a good start might involve a "census" detailing what kinds of codelets are present, and for each type, giving its percentage of the total urgency. For the cytoplasm, listing the top-level gloms' appearances (letter sequences) might be a good beginning. That way, periodic and quasi-periodic assembly and disassembly of structures would be easily detectable. But again, the tricky problem of self-reference arises: cytoplasm-watchers and coderack-watchers must themselves be codelets, vying like all codelets for time!

These twisty ideas are not yet implemented. Clearly they will be significant in determining how Jumbo tunes itself and thus controls itself dynamically. Such feedback mechanisms offer the potential for great flexibility and power, but they are also very tricky. They are reminiscent, in ways, of human consciousness, an alluring mystery towards which AI is slowly moving [Hofstadter 1979, 1981a, 1982d, Smith 1982]. Mechanisms for self-watching are therefore something to look forward to in the developing architecture of Jumbo.

A SELF-SENSITIVE, SELF-DRIVEN SYSTEM

In Jumbo, there is no top-level "driver" making decisions about what to do next. The flow is a function of weights and random numbers, not of careful consideration and reflection at each microscopic branch-point. This makes Jumbo sound stupider than it is. In life, you often must make a decision randomly in order not to get bogged down in an infinite regress. At some level, reflection must give way to reflex. But this need not be disastrous, provided your reflexes are well-tailored ones. That is the purpose of the weights -- urgencies -- attached to codelets, and of the measures of happiness.

Together, urgencies and happinesses guide Jumbo's processing. Clearly, the flow of control, to the extent that it is determinate, is determined by urgency values. What determines those values? Some processes, such as spark-generating codelets, have fixed urgencies. Others, such as codelets for spark-evaluating, flash-evaluating, and bonding, have variable urgencies. The variation comes from the perceived mutual suitedness of the gloms involved. This in turn is determined by the INTRINSIC AFFINITY of the structures involved -- a fixed feature of English -- and their HAPPINESS. Thus happinesses and affinities determine the urgency values for bonding- and disbanding-codelets.

In this way, happinesses and urgencies are intertwined, and they combine to yield a total outcome that is the overall, observable behavior of the system. Jumbo thus emerges as an exquisitely self-sensitive collection of processes: a coherent whole composed of many independent interwoven strands. (See the "Prelude...Ant Fugue" and comments in [Hofstadter 1979] or [Hofstadter 81a]. See also [Rumelhart 1982], [Hinton 1981], especially Chapter 1, and [Feldman 1982].)

EPILOGUE: JUMBO'S (EPI)PHENOMENAL INTELLIGENCE

Jumbo's intelligence, if indeed it has any, clearly has not been directly programmed; rather, it emerges as a statistical consequence of the way that many small things that one HAS designed interact with each other. It is like the chess program that "likes to get its queen out early", surprising its programmer, who never explicitly put such a strategic concept into the program. Daniel Dennett calls this an "innocently emergent" quality [Dennett 1977]; I call it an "epiphenomenon" [Hofstadter 1979, 1982c, 1982d]. One could therefore say that Jumbo involves a third metaphor: that of statistical mechanics, which explains how macroscopic order can emerge from microscopic chaos.

This view of intelligence goes against the grain of traditional AI work, which seeks to find explicit (nonstatistical) rules governing the flow of thoughts. The hope is to describe thoughts at their own level without having to resort to describing biological underpinnings of them. It is an admirable dream, but it is not a priori obvious that such laws exist.

I call this the "Boolean Dream" after George Boole who set forth an elegant calculus called "The Laws of Thought" [Boole 1855], which we know as Boolean algebra. Is the Boolean dream realizable? Most AI people seem to hope so; some even feel that a claim to the contrary is an antiscientific claim. I see things differently. My view is that AI and the Boolean dream are closely related but not synonymous. Thus the perishing of the Boolean dream would not be a setback to AI, but simply a recognition that some high-level mental phenomena are innocently emergent. The goal of AI would then shift from trying to capture the "laws of thought" in an elegant mathematical formalism to trying to understand the laws governing subcognitive events (let us call such events "mentals") that collectively go to make up cognition, or the dynamics of thinking.

To sum matters up, then, we could say that the shifted burden of AI should eventually become the clarification of the meaning of the following corny but hopefully catchy "equation":

thinkodynamics = statistical mentals.

O, a noble dream!

REFERENCES

- Arnold, Henri and Bob Lee. "Jumble #21". Signet (New American Library), 1982.
- Bergerson, Howard. "Palindromes and Anagrams". Dover, 1973.
- Bobrow, Daniel G. and Bertram Raphael. "New Programming Languages for AI Research". ACM Computing Surveys, Vol. 6, No. 3, September 1974.
- Bongard, Mikhail. "Pattern Recognition". Hayden Book Co. (Spartan Books), 1970.
- Boole, George. "The Laws of Thought" (1855). Dover.
- Dennett, Daniel C. "Brainstorms". Bradford, 1977.
- Erman, Lee D., Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty". ACM Computing Surveys, Vol. 12, No. 2, June 1980, pp. 213-253.
- Feldman, J. A. and D. H. Ballard. "Connectionist Models and Their Properties". Cognitive Science, Vol. 6, No. 3, July-September, 1982, pp. 205-254.
- Hanson, A. and E. Riseman, eds. "Computer Vision Systems". Academic Press, 1978.
- Hinton, Geoffrey E. and James A. Anderson, eds. "Parallel Models of Associative Memory". Lawrence Erlbaum Associates, 1981.
- Hofstadter, Douglas R. "Gödel, Escher, Bach: an Eternal Golden Braid". Basic Books, 1979.
- Hofstadter, Douglas R. and Daniel C. Dennett, eds. "The Mind's I". Basic Books, 1981.
- Hofstadter, Douglas R. "Metamagical Themes: Roles and Analogies in Human and Machine Thought". Scientific American, Vol. 245, No. 3, September, 1981, pp. 18-30.
- Hofstadter, Douglas R. "Metamagical Themes: On Sphexishness, 'Jootsing', and Creativity". Scientific American, Vol. 247, No. 3, September, 1982, pp. 18-34.
- Hofstadter, Douglas R. "Metamagical Themes: On Variations on a Theme, Slippability, and Creativity". Scientific American, Vol. 247, No. 4, October, 1982, pp. 20-29.
- Hofstadter, Douglas R. "Artificial Intelligence: Subcognition as Computation", in F. Machlup and Una Mansfield (eds.), "The Study of Information", to be published in 1983 by John Wiley and Sons. Also Indiana University Computer Science Department Technical Report #132, November, 1982.

- Hofstadter, Douglas R. "Who Shoves Whom Around Inside the Careenium?" *Synthese*, Vol. 53, No. 2, November, 1982, pp. 189-218. Also Indiana University Computer Science Department Technical Report #130, July 1982.
- Hofstadter, Douglas R. "Metafont, Metamathematics, and Metaphysics: Comments on Donald Knuth's Article 'The Concept of a Meta-Font'". *Visible Language*, Vol. XVI, No. 4, Autumn, 1982, pp. 309-338. Also Indiana University Computer Science Department Technical Report #136, December, 1982.
- Hofstadter, Douglas R. "On Seeking Whence", forthcoming Indiana University Computer Science Department Technical Report, 1983.
- Holland, John H. "Genetic Algorithms and Adaptation". University of Michigan Department of Behavioral Science Technical Report #34, December 1981.
- Lea, W. A., ed. "Trends in Speech Recognition". Prentice-Hall, 1980.
- Lehninger, Albert. "Biochemistry" (2nd edition). Worth Publishers, 1975.
- Reddy, D. Raj, et al. "Working Papers in Speech Recognition, IV: The HEARSAY II System". Carnegie-Mellon University Computer Science Department Technical Report, February, 1976.
- Rumelhart, David E. and Donald A. Norman. "Simulating a Skilled Typist: A Study of Skilled Cognitive-Motor Performance". *Cognitive Science*, Vol. 6, No. 1, January-March, 1982, pp. 1-36.
- Schank, Roger C. "Language and Memory". *Cognitive Science*, Vol. 4, No. 3, July-September, 1980, pp. 243-284.
- Simon, Herbert A. "Studying Human Intelligence by Creating Artificial Intelligence" (1980 Procter Lecture). *American Scientist*, Vol. 69, No. 3, May-June, 1981, pp. 300-309.
- Smith, Brian C. "Reflection and Semantics in a Procedural Language". Massachusetts Institute of Technology, Laboratory for Computer Science Technical Report #272, 1982.
- Smolensky, Paul. "Harmony Theory: A Mathematical Framework for Learning and Parallel Computation". Unpublished manuscript, Center for Human Information Processing, University of California at San Diego, 1983.
- Waterman, D. A. and Frederick Hayes-Roth, eds. "Pattern-Directed Inference Systems". Academic Press, 1978.

AN APPROACH TO LEARNING FROM OBSERVATION

Gerald DeJong
 Coordinated Science Laboratory
 University of Illinois at Urbana-Champaign
 1101 W. Springfield Avenue
 Urbana, IL 61801

ABSTRACT

Learning from observation requires a system to appreciate the significance of an event (or set of events) which are either fortuitous or part of another's planning process. Then the new events must be generalized into a new concept. We present a framework in which these processes can occur. The approach is called explanatory schema acquisition. It involves knowledge-based generalization which can construct a first-pass generalized concept from just one input example.

Key terms: Schema learning, machine learning, knowledge acquisition, artificial intelligence, natural language processing, induction.

I. INTRODUCTION

There are two problems that the generalization process must face. The first is to know when it should be applied. Clearly, every input text ought not to cause the system to construct a new schema. Only "interesting" inputs should invoke the schema acquisition system. The second problem is how to perform the generalization. There are a number of subproblems here, for example, selecting which events and objects should be generalized, imposing limits on the extent of generalization, and actually carrying out the schema construction.

Briefly, the approach can be summarized as 1) understanding the input, 2) evaluating the five generalization criteria outlined above, and 3) possibly generalizing the input to a new schema based on the evaluations of (2). The understanding phase is not significantly different from most planning-type understanders [Schmidt & Sridharan, 1977; Cullingford, 1978; DeJong, 1979; Kolodner, 1980; Lebowitz, 1980] with one exception. The understander maintains data dependency supports [Fikes, 1975; Doyle, 1979] for all of the inferences necessary to make sense of the input. This dependency network drives the generalization process. The particular input representation is generalized as far as possible without violating the dependency support structure. Violating the dependency support structure would undermine the reasons the system had in believing the story

makes sense. Usually, input can be generalized in a number of different ways. The particular generalization selected is determined by fitting the input into a taxonomy of generalization techniques.

II. TO GENERALIZE OR NOT TO GENERALIZE

There are five aspects to be considered when deciding whether or not to generalize an input into a new schema. By hypothesis we will assume that the input did not match an existing schema (if it had then the system already possesses the desired schema and indeed that schema would have been used to process the story). If any of these five conditions does not hold, constructing a new schema from this input is inappropriate.

The criteria are:

- 1) Is the main goal of a character achieved?
- 2) Is the goal a general one?
- 3) Are the resources required by the goal achiever generally available?
- 4) Is this new method of achieving the goal at least as effective as the other known volitional schemata to achieve this goal?
- 5) Does the input match one of the known generalizable patterns?

These criteria are tested for all goals in the story. The first criterion "Was the goal achieved?" is self explanatory and easily judged. The second "Is it a general goal?" and the third "Are the resources generally available?" require some discussion.

Novelty alone in an approach to achieving a goal is not sufficient to warrant constructing a new schema. Consider, for example, a plot from the "Mission Impossible" television series. These plots are noteworthy in that they are very novel. They all use bizarre methods to achieve rather peculiar goals. Furthermore, they are always successful. However, the goals achieved are not the type that arise in ordinary life and the resources and skills needed are so specialized and uncommon that the same solution would never be applicable again.

Clearly a new schema should be constructed only if there is a reasonable expectation that it will be helpful in future processing. If a schema will never be used again, it should not be constructed in the first place.

This work was supported in part by the Air Force Office of Scientific Research under grant F49620-82-K-0009 and in part by the National Science Foundation under grant NSF IST 81-20254.

How can the utility of a particular goal be judged? The answer to this is closely tied to where goals come from. Achieving a goal which arises from general conditions important to an individual's well-being and using readily available resources is likely to result in an interesting new schema, one which will arise again and again. For the solution, we use an aspect of Schank & Abelson's theory of planning [Schank & Abelson, 1977]. In their view themes give rise to the highest level goals (goals which are not simply subgoals in the achievement of other goals). Interpersonal and life themes are what we are interested in. An example of the former is a husband offering (and therefore, at some level, wanting) to type a term paper for his over-worked student wife. This exemplifies the theme of taking on goals of others that one cares for. It is not necessary to go beyond this theme in explaining the husband's actions. Examples of life themes are attempting to satisfy one's hunger, to gain money, or to relieve boredom. Themes give rise to goals that require no further justification.

The fourth criterion is self-explanatory. The idea is that the system should not bother constructing schemata that are much less efficient than similar already-known schemata. In a natural language input this would occur only if a character were using a highly sub-optimal plan.

The fifth criterion is that the input must be identified within the taxonomy of generalization techniques. This is discussed in the next section.

III. TAXONOMY OF GENERALIZATIONS

There are four situations which when recognized in the text either individually or in combination ought to invoke the generalization routines. They are:

- Schema Composition
- Secondary Effect Elevation
- Schema Alteration
- Volitionalization

In the first part of this section we will illustrate each of these situations with an example.

3.1. Schema composition

The first situation we will discuss is called schema composition. Basically, it involves composing known schemata in a novel way. Typically, this will involve a primary schema, essentially unchanged, with one or more of its preconditions satisfied in a novel way by other known schemata.

To clarify the procedure, consider an example. This example is a story about a kidnapping. Let us assume that we, the readers of this example, do not yet have a schema for kidnapping or extortion or any similar notion. We do, however, assume the knowledge of a considerable quantity of background information about stealing, bargaining, the use of normal physical objects, and goals of people and institutions.

Paris police disclosed Tuesday that a man who identified himself as Jean Maraneaux

abducted the 12 year old daughter of wealthy Parisian businessman Michel Boullard late last week. Boullard received a letter containing a snapshot of the kidnapped girl. The next day he received a telegram demanding that 1 million francs be left in a lobby waste basket of the crowded Pompidou Center in exchange for the girl. Asking that the police not intervene, Boullard arranged for the delivery of the money. His daughter was found wandering blindfolded with her hands bound near his downtown office on Monday.

In this story the primary schema is BARGAIN, a schema which we assumed the system already knew. One of the preconditions specified in the BARGAIN schema is that each party to the bargain must convince the other that he can indeed deliver his side of the bargain. For Maraneaux, this corresponds to making Boullard believe that he (Maraneaux) has control of Boullard's daughter and can, therefore, relinquish the girl to him. Maraneaux achieves this by actually establishing control over the daughter (via an instance of the STEAL schema) and then sending Boullard a photograph. To the system, this is a novel way to satisfy BARGAIN's preconditions.

3.2. Secondary effect elevation

Consider the following scenario:

Fred wanted to date only Sue, but Sue steadfastly refused his overtures. Fred was on the verge of giving up when he saw what happened to his friend, John: John wanted to date Mary but she also refused. John started seeing Wilma. Mary became jealous and the next time he asked her, Mary eagerly accepted. Fred told Sue that he was going to make a date with Lisa.

Here Fred has not acquired a new schema; he has used an existing schema (DATE) in a new way. This is called secondary effect elevation. Fred's DATE schema already contains all of the knowledge necessary for resolving his dilemma. The problem is that the normal DATE schema is organized in the wrong way. In secondary effect elevation situations an existing schema is annotated indicating that the schema may be used to achieve a result which is normally neutral or negative.

The main purpose of the DATE schema is to satisfy certain recurring social goals (like companionship, sex, etc.). DATE contains secondary effects as well. These are often undesirable effects accompanying the main, planned effects. For example, one is usually monetarily poorer after a date. Another secondary effect is that if one has an old girlfriend, she may become jealous of a new date.

What Fred learned from John's experience is that it is occasionally useful to invoke the DATE schema in order to cause one of its secondary effects (jealousy) while completely ignoring the usual main goal.

Just as with schema composition, the existing schema is changed to reflect a generalization made from a specific instance. In this case, the

specific instance is John's interactions with Mary. Notice, however, that Fred did not simply copy John's actions. John actually made a date with Wilma while Fred only expressed an intention to date Lisa. This is not an earth-shaking difference, but in the context of dating it is extremely significant. In the normal DATE situation expressing an intention to date someone is not nearly so satisfying as an actual date. Once modified for the purpose of causing jealousy, however, expressing an intention for a date and actually carrying it out can be equally effective.

One might argue that the distinction between main and secondary effects of a schema is otiose and, in situations such as this, even deleterious. After all, DATE already had all of the information necessary for solving Fred's problem. If a system simply treats all of the effects of a schema the same, then any effect can be singled out during the planning process to be used as the main goal. There is, however, a strong argument against this position. The possible desired effects of a schema do not exist only within the schema itself. They are used to organize and select among schemata in both understanding and planning applications. Many effects (like feeling more tired after a date than before) will not be used in the normal planning or understanding process. If they are treated the same as legitimate main goals the system will be swamped in a combinatorial quagmire of undifferentiated possibilities, most of which are wildly implausible. For example, we do not want our understanding process to predict that John will take a nap when it is told that John dated Mary. Given the input "John took a nap" the system ought to be able to justify it. However, it ought not actively predict it. Given the multiplicity of individual actions making up the DATE schema (each with its own set of effects) the vast majority of the effects from this schema (and any other schema) are simply irrelevant to overall planning and understanding processes. Instead, we would like our system to single out the plausible volitional effects of its schemata and use only those for schema organization and selection. Thus, in our example, Fred has constructed, via secondary effect elevation, a new use of the DATE schema.

3.3. Schema alteration

Schema alteration involves modifying a nearly correct schema so that it fits the requirements of a new situation. The alteration process is guided by the system's world model. This is illustrated by the following brief anecdote:

I had occasion to replace temporarily a broken window in my back door with a plywood panel. The plywood sheet from which the panel was to be cut had a "good" side and a "bad" side (as does most raw lumber). The good side was reasonably smooth while the bad side had several ruts and knot holes. I automatically examined both sides of the sheet (presumably as part of my SAWING or CUTTING-A-BOARD-TO-FIT schema) and selected the good side to face into the house with the bad side to be exposed to the elements. After I had

cut the panel and fitted it in place I noticed that several splinters had been torn out leaving ruts in the "good" side. I immediately saw the problem. Hand saws only cut in one direction. With hand saws, the downward motion does the cutting while the upward motion only repositions the cutting blade for another downward motion. I had cut the wood panel with the "good" side facing down. The downward cutting action has a tendency to tear splinters of wood out of the lower surface of the board. Since the good side was the lower surface, it suffered the loss of splinters. If I had to perform the same action again, I would not make the same mistake. I would cut the board with the good side facing up. However, what I learned was not just a simple specialized patch to handle this particular instance of splintering. Since I knew the cause of the splintering, I knew that it would not always be a problem: it is only a problem when 1) the lumber is prone to splintering, 2) there is a "good" side of the board that is to be preserved, and 3) one is making a crosscut (across the wood's grain) rather than a rip cut (along the grain). Moreover, the solution is not always to position the wood with the good side up. My electric saber saw (also a reciprocating saw) cuts during the upward blade motion rather than the downward motion. Clearly, the solution when using the saber saw is the opposite: to position the board with the good side down. Now, these are not hard and fast rules: with a sufficiently poor quality sheet of plywood splintering would likely always be a problem. Rather, these are useful heuristics that lead to a refinement of the SAWING schema.

Note that this refinement to the SAWING schema is far more general than required to handle the particular problem that gave rise to it. The refinement contains contingencies relevant to the use of saber saws even though no saber saw was used in the immediate problem. This is possible because the refinement is driven by world model, not just the problem. The SAWING schema was altered by identifying and eliminating the offending cause in the underlying knowledge-based explanation of the phenomena.

3.4. Volitionalization

This situation involves transforming a schema for which there is no planner (like VEHICLE-ACCIDENT, ROULETTE, etc.) into a schema which can be used by a planner to attain a specific goal. Consider the following story:

Herman, who was married to Joy for 15 years had fallen in love with his secretary, Heather. When Joy refused to divorce him, Herman cut a hydraulic brake hose in Joy's car. The next time she drove down the winding road to town she lost control of her car and struck a tree. The car burst into flames.

This story describes a vehicle accident. However, unlike most vehicle accidents, this one has an active agent. The VEHICLE-ACCIDENT schema is normally non-volitional. That is, it dictates what to expect in a vehicle accident situation, but it does not allow planned invocation. This schema cannot be involved in problem-solving planning.

The story illustrates that an event which was previously only attributable to accidental causes can, in fact, be controlled. Thus, a volitional counterpart can be constructed for the previously non-volitional schema. This new volitional schema has certain effects that can be labeled as its main goals, new constraints on the actors and objects, etc. It is, in fact, a new schema in its own right. Furthermore, using the underlying explanation of the story to drive the generalization process, the schema can encompass situations significantly different from the one described in the story. For example, it should handle a story about a student weakening the steering rod of a professor's car after receiving a failing grade.

IV. AN EXAMPLE

Here, we give an example of the first-pass implementation of the system. It illustrates a new schema acquired by volitionalization. The system does not have a natural language front end. The inputs are given in conceptual form. These are denoted by lines starting with "Processing---". There are just four inputs. English versions of the inputs are given below. For readability, this output is a slightly altered form of the program's output. The system generates many more GENSYM atoms and refers to them by their GENSYM names. The GENSYMs have been replaced with mnemonic symbols.

ENGLISHIFIED INPUT:

- 1) CLAUDIUS OWNED AN ISLAND ESTATE.
- 2) AGRIPPINA FED CLAUDIUS POISONED MUSHROOMS.
- 3) CLAUDIUS DIED.
- 4) AGRIPPINA INHERITED THE ISLAND ESTATE.

The first input

Processing---(\$POSS (ACTOR *CLAUDIUS*)
(OBJECT *ESTATE*))

New Schema---S00001 (\$POSS (ACTOR *CLAUDIUS*)
(OBJECT *ESTATE*))

Setting bindings and links

New Schema---S0000 (\$DESIRE-FOR-MATERIAL-THINGS
(ACTOR *CLAUDIUS*) (OBJECT *ESTATE*))
Setting bindings and links

The second input

Processing---(\$FEED (ACTOR *AGRIPPINA*)
(OBJECT *POISON* TYPE *MUSHROOM*))

New Schema---S00003 (\$FEED (ACTOR *AGRIPPINA*)
(OBJECT *POISON*) (TO *CLAUDIUS*))

Setting bindings and links

New Schema---S00005 (\$EAT (ACTOR *CLAUDIUS*)
(OBJECT *POISON*))

Activated by S00003

Setting bindings and links

\$EATING poison brings in the poison schema. We call it NAIVE POISON to remind ourselves that it is an incomplete schema containing little more than what is needed for this story.

New Schema---S00006 (\$NAIVE-POISON (ACTOR
AGRIPPINA) (INSTRUMENT *MUSHROOM*)
(OBJECT *CLAUDIUS*))

Activated by S00003

Setting bindings and links

New Schema---S00007 (\$MOVE (ACTOR *AGRIPPINA*)
(OBJECT *MUSHROOM*) (FROM (OUTSIDE *CLAUDIUS*)
(TO (INSIDE *CLAUDIUS*)))))

Activated by S00006

Setting bindings and links

New Schema---S00008 (\$BODILY-HARM (OBJECT
CLAUDIUS))

Activated by S00006

Setting bindings and links

New Schema---S00009 (\$HEALTH (CREATURE
CLAUDIUS) (STATE (*VAR* S1)))

Activated by S00008

Setting bindings and links

New Schema---S00010 (\$INGEST (ACTOR *AGRIPPINA*)
(OBJECT *POISON*) (TO *CLAUDIUS*))

Activated by S00006

Setting bindings and links

This \$DEATH is predicted by the poisoning

New Schema---S00011 (\$DEATH (OBJECT *CLAUDIUS*))

Activated by S00006

Setting bindings and links

Old Schema---S00009 (\$HEALTH (CREATURE
CLAUDIUS) (STATE *ALIVE*))

Activated by S00011

Setting bindings and links

New Schema---S00012 (\$HEALTH (CREATURE
CLAUDIUS) (STATE *DEAD*))

Activated by S00011

Setting bindings and links

Alternatively, Claudius might only become ill.

This is also predicted.

New Schema---S00013 (\$NAUSEA (ACTOR *CLAUDIUS*))

Activated by S00006

Setting bindings and links

If Claudius dies, Agrippina might have planned his death.

New Schema---S00014 (\$NAIVE-REMEDITATED-MURDER
(ACTOR *AGRIPPINA*) (OBJECT *CLAUDIUS*))

Activated by S00006

Setting bindings and links

Old Schema---S00011 (\$DEATH (OBJECT *CLAUDIUS*))

Activated by S00014

Setting bindings and links

Old Schema---S00006 (\$NAIVE-POISON (ACTOR
AGRIPPINA) (INSTRUMENT *MUSHROOM*) (OBJECT
CLAUDIUS))

Activated by S00014

Setting bindings and links

New Conjunction Node---S00015 (AND (\$DEATH
 (ACTOR *CLAUDIUS*)) (NOT (\$PUNISH (OBJECT
 AGRIPPINA))))

Old Schema---S00011 (\$DEATH (OBJECT *CLAUDIUS*))

The third input
 Processing---(\$DEATH (OBJECT *CLAUDIUS*))
 Old Schema---S00011 (\$DEATH (OBJECT *CLAUDIUS*))

This matches the dying prediction. Thus, very
 little processing must be done.

The fourth input
 Processing---(\$NAIVE-INDIVIDUAL-INHERIT
 (HEIR *AGRIPPINA*) (BEQUESTS *ESTATE*))

New Schema---S00018 (\$NAIVE-INDIVIDUAL-INHERIT
 (HEIR *AGRIPPINA*) (BEQUESTS *ESTATE*))
 Setting bindings and links

Old Schema---S00001 (\$POSS (ACTOR *CLAUDIUS*)
 (OBJECT *ESTATE*))
 Activated by S00018
 Setting bindings and links

Old Schema---S00011 (\$DEATH (OBJECT *CLAUDIUS*))
 Activated by S00018
 Setting bindings and links

New Schema---S00021 (\$HEALTH (CREATURE
 AGRIPPINA) (STATE *ALIVE*))
 Activated by S00018
 Setting bindings and links

Old Schema---S00012 (\$HEALTH (CREATURE
 CLAUDIUS) (STATE *DEAD*))
 Activated by S00018
 Setting bindings and links

New Schema---S00022 (\$TRANSFER-OF-POSSESSION
 (FROM *CLAUDIUS*) (TO *AGRIPPINA*) (OBJECT
 ESTATE))
 Activated by S00018
 Setting bindings and links

Old Schema---S00001 (\$POSS (ACTOR *CLAUDIUS*)
 (OBJECT *ESTATE*))
 Activated by S00022
 Setting bindings and links

New Schema---S00025 (\$POSS (ACTOR *AGRIPPINA*)
 (OBJECT *ESTATE*))
 Activated by S00022
 Setting bindings and links

Finally, the system has a motivation for the
 murder.

New Schema---S00026 (\$MATERIAL-DESIRE (ACTOR
 AGRIPPINA) (OBJECT *ESTATE*))
 Activated by S00025
 Setting bindings and links

VOLITIONALIZATION TRIGGERED

NEW SCHEMA V00035:
 (FORM SCHEMA
 (VARS (BENEFACITOR (*VAR* V00031)
 PRIOR-HEIRS (*VAR* V00033)
 BEQUESTS (*VAR* V00029)
 HEIR (*VAR* V00030)))

ACTOR (*VAR* V00030)
 OBJECT (*VAR* V00031))
 (POINT-OF-VIEW (*VAR* V00030))
 (GOAL V00032)
 (ACTIVATE
 (V00032 (\$POSS (ACTOR (*VAR* V00030))
 (OBJECT (*VAR* V00029))))
 V00033
 (\$NAIVE-INDIVIDUAL-INHERIT
 (BENEFACITOR (*VAR* V00031))
 PRIOR-HEIRS (*VAR* V00033)
 BEQUESTS (*VAR* V00029)
 HEIR (*VAR* V00030))
 V00034
 (\$NAIVE-PREMEDITATED-MURDER ACTOR
 (*VAR* V00030)
 OBJECT (*VAR* V00031))
 (COMPLEX (V00033 V00034))))

V. CONCLUSION

The example shows a new schema acquired by the system. The form of this schema is identical to those pre-programmed into the system. The only difference is in the mnemonic naming of variables. There are several problems with this implementation which are the subject of current research. First, there is an inadequate treatment of time. The system assumes (whenever it cares) that the inputs are presented in the same order as the events they report. The system performs (through schema expansion) a good deal of causal analysis. This also implies some temporal orderings. However, by and large, difficult time problems do not surface for this simple story. A second problem is with variable constraints. This implementation does very little reasoning about objects. Most of its generalization is performed on constituent events rather than the objects participating in those events. These generalizations (such as generalizing POISON to MURDER) seemed less obvious and therefore more interesting. Furthermore, there has been previous work on generalizing objects [Fikes, 1972]. Finally, there is a good deal of work to be done on goal manipulation of story characters and on schema representation. The representations used here are ad hoc and cannot easily handle the next task we have chosen (a sequence of kidnapping stories).

Unlike more correlational approaches [Michalski, 1977; Fox & Reddy, 1977; Langley, 1981], the procedure is capable of constructing a schema from one input example. Nor is the procedure "failure-driven" [Kolodner, 1980; Lebowitz, 1980; Schank, 1982]. Nor is it primarily analogical [Winston, 1980; Carbonell, 1982]. The LEX system [Mitchell, Utgoff & Banerji, 1983] is also a significantly different approach. In that system concepts are acquired that are judged to be useful through search and experimentation not through an analysis of why an operator was useful. In explanatory schema acquisition concepts are formed entirely on the basis of this kind of analysis. The knowledge-based concept approach owes much to Soloway [1977] and Lenat [1976].

ACKNOWLEDGMENT

I am indebted to the members of the CSL learning group: Chris Debrunner, Paul Harrington,

Paul O'Rorke, and Alberto Segre. Paul O'Rorke implemented the system that performed the example.

REFERENCES

- Carbonell, J., "Experiential Learning in Analogical Problem Solving," Proc. Nat'l. Conf. on Artificial Intelligence, Pittsburgh, PA, 1982.
- Cullingford, R., "Script Application: Computer Understanding of Newspaper Stories," Research Rept. 116, Yale Computer Science Dept., New Haven, CT, 1978.
- DeJong, G., "Prediction and Substantiation: A New Approach to Natural Language Processing," Cognitive Science, vol. 3, pp. 251-273, 1979.
- Doyle, J., "A Truth Maintenance System," Artificial Intelligence, vol. 12, no. 2, pp. 231-272, 1979.
- Fikes, R., "Deductive Retrieval Mechanisms for State Description Models," Proc. 4th IJCAI, pp. 99-106, 1975.
- Fikes, R., Hart, P. and Nilsson, N., "Learning and Executing Generalized Robot Plans," Artificial Intelligence, vol. 3 no. 2, pp. 251-288, 1972.
- Fox, M. and Reddy, R., "Knowledge-Guided Learning of Structural Descriptions," Proc. 5th IJCAI, Cambridge, MA, 1977.
- Kolodner, J., "Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model," Computer Science Research Rept. 187, Ph.D. dissertation, Yale Univ., New Haven, CT, 1980.
- Langley, P., "Data-Driven Discovery of Physical Laws," Cognitive Science, vol. 5, pp. 31-54, 1981.
- Lebowitz, M., "Generalization and Memory in an Integrated Understanding System," Computer Science Research Rept. 186, Ph.D. dissertation, Yale Univ., New Haven, CT, 1980.
- Lenat, D., "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search," AIM-286, AI Laboratory, Stanford Univ., Stanford, CA, 1976.
- Michalski, R., "A system of Programs for Computer Aided Induction: A Summary," Proc. 5th IJCAI, Cambridge, MA, pp. 319-320, 1977.
- Mitchell, T., Utgoff, P. and Banerji, R., "Learning by Experimentation: Acquiring and Refining Problem Solving Heuristics," in Machine Learning, R. Michalski, J. Carbonell, & T. Mitchell (Eds.), Tioga, Palo Alto, CA, 1983.
- Schank, R., Dynamic Memory, Cambridge Univ. Press, Cambridge, 1982.
- Schank, R. and Abelson, R., Scripts Plans Goals and Understanding, Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.
- Schmidt, C. and Sridharan, N., "Plan Recognition Using a Hypothesize and Revise Paradigm: An Example," Proc. 5th IJCAI, Cambridge, MA, pp. 480-486, 1977.
- Soloway, E., "Knowledge Directed Learning Using Multiple Levels of Description," Ph.D. dissertation, Computer Science Dept., Univ. of Massachusetts, Amherst, MA, 1977.
- Winston, P., "Learning and Reasoning by Analogy," Commun. ACM, vol. 23, no. 12, 1980.

CONCEPT LEARNING IN A RICH INPUT DOMAIN¹

Michael Lebowitz

Columbia University, Department of Computer Science
New York, NY 10027

ABSTRACT

Automatic concept learning from large amounts of complex input data is an interesting and difficult process. In this paper we discuss how the use of a permanent, generalization-based, memory can serve as an important tool in developing programs that learn in rich input domains. Its use allows programs to determine what concepts to learn, as well as definitions of the concepts. We present in this paper a characterization of our research, and then describe our use of Generalization-Based Memory in two programs under development at Columbia, UNIMEM and RESEARCHER, and describe how they perform concept evaluation and generalization of complex structural descriptions, problems typical of those we are concerned with.

Key Terms: Learning, automatic concept formation, generalization, Generalization-Based Memory, intelligent information systems, artificial intelligence, cognitive science.

1 INTRODUCTION

Automatic concept learning in the form of generalization has been shown to be useful in interpreting and organizing large amounts of information about a domain [Lebowitz 80; Schank 82; Lebowitz 83a], as well as being an interesting task in its own right. Recently, we have been concerned with the development of new methods of concept formation that make use of permanent memory and include the determination of what concepts to learn, as well as the definitions of the concepts. In particular, we have concentrated on the problems of concept formation from a stream of input that is complex in any of several different ways. In this paper, we will detail the class of problems we are addressing, present the basic learning technique that we use, known as Generalization-Based Memory (GBM), and indicate solutions to some of the specific problems that are involved.

Much of the concept learning research that has been done in Artificial Intelligence has consisted of either supplying programs with examples, and possibly counter-examples, of specified concepts and having these

programs determine definitions of those concepts ([Winston 72; Mitchell 78], for example) or of using largely analytic techniques to classify input (e.g., [Michalski 80; Langley 81]). In "real-world" settings, the crucial concepts to be learned -- those that best help explain and organize information about a domain -- are not pre-supplied; rather, it is necessary to determine these concepts from a stream of very complex input data. Consequently, our research concentrates not just on how to compare examples, but also on methods for determining what examples to compare, which largely determines what concepts to create.

To take examples from various programs we have worked on, we will look here at how intelligent systems could extract from complex input streams generalizations such as: "States that have high school expenditures have high per capita incomes" (from information about the states of the United States); "A large class of disk drives use flexible (floppy) discs" (from patent abstracts about disk drives); or "Terrorist attacks in Northern Ireland are frequently carried out by the IRA" (from news stories about terrorism), to the same extent that people do.

We describe here a powerful memory organization and concept learning technique, Generalization-Based Memory. GBM was developed for the computer program IPP [Lebowitz 80; Lebowitz 83a], based on intuitions about how complex human episodes might be stored in memory in a manner analogous to Schank's MOPs [Schank 82; Kolodner 80]. We will discuss the advantages gained by using the same technique in a more traditional concept learning environment or for intelligence information systems that involve complex streams of input. Our presentation of the problems of concept learning from complex input is done in the context of two intelligent information systems being developed at Columbia, UNIMEM and RESEARCHER, both of which use GBM.

UNIMEM is a program that can accept a large quantity of relatively unstructured facts about a domain, use generalization techniques to determine important concepts, and use these concepts to organize the information in a fashion that allows further generalization and intelligent question answering. For example, if information about the states in the U.S. was given to such a program (a domain used in prototype testing), the program might determine that New England states, or states with large education budgets were useful concepts. UNIMEM is being used to study problems that can arise when the individual items used for learning are not highly structured, each consisting simply of a set of descriptive features.

¹This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-82-C-0427. Comments by Kathy McKeown on an earlier draft of this paper were most helpful.

The problems in forming concepts from complex input data involved in our research with UNIMEM include: the impact of domain-dependent knowledge on concept learning; categorizing numeric input information so that generalization is possible; concept evaluation and refinement from further examples; using concepts that very slightly contradict new input items; dealing with concepts that change over time; and question answering based on Generalization-Based Memory. In this paper we present the basic techniques for using GBM and for evaluating concepts in the context of UNIMEM.

RESEARCHER [Lebowitz 83b], in contrast with UNIMEM, deals with highly structured, physical descriptions of devices. RESEARCHER reads patent abstracts in natural language form, and then remembers and generalizes information from these texts. Understanding patents tends to require complex physical descriptions of objects. In this paper, we use RESEARCHER as a context in which to discuss the problems of comparing complex, highly structured representations.

Figure 1 shows some typical concepts generalized by each of the Generalization-Based Memory programs mentioned here. The IPP and UNIMEM generalizations were actually made by the programs (although the English was generated by hand), and the RESEARCHER examples show what we are aiming for.

IPP Concepts:

Bombings in El Salvador cause damage, but do not often hurt anyone.
Urban terrorists in Italy frequently use silencer equipped pistols.

UNIMEM Concepts:

State class -- High urban percentage, low minority percentage, moderate income, low taxes, manufacturing important [RI, NJ, TX, MI, FLA, OH]

State class -- High value of farmland, fairly high population, manufacturing, agriculture, tourism important [NC, ARK, TENN, MINN, WISC, VA, MO]

RESEARCHER Concepts:

Floppy disk drive

Double density disk drive

Fully enclosed disk drive

Figure 1: GBM Concept Examples

In the remainder of this paper, we describe how our research relates to other work in concept formation, and present an overview of our concept learning methods, concentrating on our use of Generalization-Based Memory. Finally, we describe the way we handle concept evaluation and generalization of complex structural descriptions, problems typical of those we are concerned with.

2 COMPLEX INPUT DOMAINS

The intelligent information systems we are developing basically engage in what [Michalski 83] calls *multiple concept learning from observation* (*descriptive generalization*), in that these programs are given large number of examples, with no pre-specification of the concepts to generalize, and they acquire sets of concepts by deciding what instances to compare and how such examples are similar. The concepts derived are often overlapping, in that many concepts can describe the same example.

The tasks of our programs also involve aspects of Michalski's *concept acquisition*. In addition to determining the properties of instances in the classes that they create they fit objects to those classes. There are elements in our programs of both observing patterns in data and developing discriminant descriptions of the classes thereby derived.

Our research is characterized by several other properties, all somewhat novel for working systems, but, we feel, crucial to the development of useful, dynamic, information systems. We look at learning that is *incremental*. It is not possible in systems that are continually receiving input to wait for all examples to be available for inspection before creating concepts. We require that, after every example is processed, our systems have made the best possible generalizations based on the input that has been processed so far.

We expect that our systems will ultimately deal with *large numbers of examples*. It is the ability to deal with many examples and many concepts simultaneously that gives human learning the power we would like our systems to have. No method that requires comparison of a new instance with all, or a large portion of, previous examples will be acceptable, for computational reasons. Even comparison with generalized concepts must be done in a principled way. We are, in addition, sometimes concerned with cases where the individual items to be generalized are themselves complex, as in RESEARCHER.

The final parameter that characterizes all our work is that we are dealing with *"pragmatic" generalizations*. That is, we are concerned with concepts that describe what is usually, but not necessarily always, true. This means, crucially, that methods that invalidate a generalized concepts on the basis of a single example are not acceptable. In the same vein, we do not require that every concept that could legitimately be generalized be found. The class of pragmatic generalizations provides more power and flexibility in representing what it is possible to learn about a rich domain.

We feel that methods for dealing with the type of input described here will be necessary in developing systems that take full advantage of the large quantities of complex information we are now able to provide them.

3 GBM

In this section, we provide an overview of the techniques used to form concepts as part of maintaining a

Generalization-Based Memory. For clarity, we describe the way the process works in UNIMEM, but the main techniques are identical in IPP and RESEARCHER.

The basic idea of Generalization-Based Memory is that a generalization begins to create a hierarchy of concepts that describe a situation from a small number of examples, and then records in memory specific items, both those that the concepts are generalized from and further examples, in terms of the generalized concepts. More specific generalizations are recorded along with specific examples under the more general cases. GBM involves identifying and defining multiple concepts, as opposed to maintaining a single model of a concept.

In order to standardize our terminology, we refer to the objects stored in memory, i.e. the input examples, which are used to build generalizations, as *instances*. In UNIMEM these are descriptions of objects in a domain. An instance is described in UNIMEM terms of a set of *features* (essentially property/value pairs). As we will see, RESEARCHER uses more complex descriptions of instances. The combinations of generalizations, themselves sets of features, and the events and sub-generalizations they organize are called *GEN-NODEs*.² GEN-NODEs form the basis of GBM. The structure of a typical GEN-NODE is shown in Figure 2.

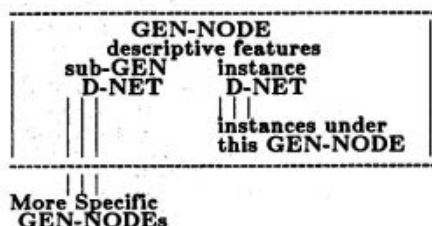


Figure 2: Schematic GEN-NODE

Generalization-Based Memory consists basically of one or more hierarchies of GEN-NODEs that describe concepts of increasing specificity. As shown in Figure 2, instances and sub-GEN-NODEs are stored under each GEN-NODE using discrimination networks (D-NETs) [Charniak, et al. 80]. (Note that a GEN-NODE can organize both instances and more specific GEN-NODEs.) D-NETs provide an efficient way to retrieve any object stored with a given set of indices. In the GBM model, every feature of an instance or sub-GEN-NODE is initially used as an index, resulting in shallow, bushy D-NETs that allow retrieval of an object given any one of its features. The resulting plethora of indices is pruned by ceasing to use as indices features that pertain to a large number of objects in a given D-NET.

The use of a hierarchy of GEN-NODEs with D-NETs as a method of memory organization allows efficient

storage of information, since information in a generalization does not have to be repeated for each instance that it describes. In addition, it allows relevant generalizations and instances -- and *only* relevant generalizations and instances -- to be efficiently found in memory during processing, allowing further generalizations. This property of GBM is largely independent of the specific knowledge representation being used.

The process of maintaining GBM, which is the learning process we are considering here, is a relatively simple one, once the memory organization method has been defined. As each new instance is processed, the most specific GEN-NODE that describes it is found. This is done easily and efficiently, using the discrimination nets that index the GEN-NODEs, starting with a very general node that covers the whole range of instances in the domain. Then, before the instance is actually indexed under that GEN-NODE, a check is made for instances already stored there that have additional features in common with the new instance, which can be found using the instance D-NET. If there are enough such features (one of many adjustable parameters of the system), a new concept is generalized, and the contributing instances indexed there. Otherwise, the new instance is simply stored under the existing GEN-NODE.³

Two further important features characterize GBM. Since concepts are generalized on the basis of few instances, they must be evaluated to eliminate over-generalization (including the elimination of whole concepts). This is discussed in Section 4. The second feature is the use of an idea known as *predictability*. While space does not permit a discussion of predictability here (see [Lebowitz 83a]), the basic idea is that only the presence of some features of a concept in an instance indicate the relevance of the concept, and that these features can be identified quite easily using GBM.

To summarize, GBM satisfies handles the domain characteristics described in Section 2 as follows:

- 1) GBM is inherently *incremental*. As each instance is added to GBM, the best possible concepts that can be generalized so far are made.
- 2) GBM is ideal for handling *large numbers of examples*. The use of a hierarchy of concepts that organize specific instances allows only instances that might lead to generalizations to be compared to each other. Relevant concepts are easily found. It is also an efficient way to store the concepts.
- 3) All concepts generalized in GBM are "*pragmatic*". No concept is removed by a single counter-example, but

²GEN-NODEs were called S-MOPs in IPP, as they are, in some sense, specialized versions of Schank's Memory Organization Packets [Schank 82].

³The process is actually a bit more complex, as a given instance can be stored in multiple spots in memory for two different reasons. An instance can either be classified initially in several different ways, each of which would indicate a place to store it, or several different "most specific" GEN-NODEs might be found, each of which would lead to the processing described in the text.

instead, the process described in the next section is used to evaluate all concepts. The generalization process is also pragmatic because it can sometimes miss concepts that could be found by comparing instances that were stored in widely different parts of GBM, but this seems a reasonable trade-off to avoid combinatoric numbers of comparisons.

4 CONCEPT EVALUATION

As mentioned in the previous section, the concept learning process we have described inherently leads to over-generalization, particularly in a domain where there is a large amount of information about each instance. Thus, we require each concept learned to be evaluated over time. For each generalization made by UNIMEM, an evaluation process continually looks for later instances for which the generalization might be relevant. This occurs as a normal part of the memory update process, since the generalizations to be evaluated are exactly those that might be used to store the new instances. UNIMEM checks whether a relevant generalization is confirmed or contradicted by each new instance.

A new instance found by UNIMEM is considered to contradict an applicable concept if it possesses a predictable feature indicating that the concept is relevant, but also another feature with the same property as the concept (such as the region of a state), but with a different value (Midwest instead of East, perhaps). When this condition occurs, confidence in the concept is undermined.

Early versions of confidence for generalizations in GBM simply involved adding or subtracting points from a numeric confidence level for each GEN-NODE, resulting in a property much like the confidence in conclusions discussed in [Collins 78], or the confidence in rule application used in some expert systems. In a domain rich in information this technique will not do, as there will almost *always* be extraneous information in each generalized concept, as the result of inevitable coincidences, that will cause confidence in the concept to be undermined.

What we would like to do when a generalization is disconfirmed is to throw away the "bad" (overly specific) parts and keep the "good" parts. The problem then reduces to identifying the components of a generalization that are overly specific, so that they can be deleted, leaving intact a valid generalization. Furthermore, for this to be useful, it must be done at a minimum of cost, hopefully occurring as a natural part of the memory update process, and requiring only a small amount of extra record-keeping.

The solution devised for UNIMEM is straightforward. Instead of keeping a single confidence level as part of each GEN-NODE, UNIMEM tracks how often each feature of a concept is confirmed or contradicted. In effect, a confidence level is maintained for each feature of each concept, rather than a single value for an entire concept.

Specifically a counter is maintained for each feature of each generalization and these counters are incremented

or decremented as their features are confirmed or contradicted, respectively, in a situation where a concept is deemed relevant. The counter modification occurs as UNIMEM determines which GEN-NODEs best describe a new instance, as described in Section 3. If a counter passes a negative threshold, then we can eliminate the feature from the generalization, since the feature has been wrong much more often than right. We sometimes have to eliminate entire generalizations when too many of their features have been eliminated. Details of this process, and an example of its application in the domain of football plays, can be found in [Lebowitz 82].

When this scheme was added to UNIMEM, it proved quite effective in culling extraneous features from generalizations, and only totally disconfirming those concepts that were completely the result of coincidence. We show here a simple example from the states domain, in which we provided UNIMEM with a number of facts about each state in the United States.

Figure 3 illustrates a concept (GND1) generalized by UNIMEM. In it, the third column of each feature (the value) indicates a category derived from numeric data by a method described in [Lebowitz 83c]. For example, the first feature of the generalization is that per capita income in the relevant states is in the third of four categories, i.e., rather high. Roughly, the concept generalized here describes states with moderately high per capita income, rather low taxes, high school expenditure, and fairly low minority population (the last is actually a broad category that covers most states). This concept can be used to describe the seven states listed.

```
GND1:
INCOME      RANGE      INC3:4
TAXES       RANGE      TAX2:5
SCHOOL-EXP  RANGE      SCH3:3
MINORITY-PCT RANGE      MW1:2
Organizing: IOWA, KANSAS, MICHIGAN, MONTANA,
             NEBRASKA, PENNSYLVANIA, TEXAS
```

Figure 3: Final UNIMEM Generalization

Figure 4 shows how this concept was initially generalized from Iowa and Nebraska (the states were presented to UNIMEM in a random order). Notice that these states are similar in a number of additional ways, e.g. they are both farm states, so UNIMEM initially generalized an over-general, and not widely applicable, concept. These extraneous features were ultimately removed by the evaluation process described in this section, leaving a much more useful concept.

```
GND1:
CRIME-RATE  RANGE      CR13:5
STATE-DEBT  RANGE      DEB2:7
INCOME      RANGE      INC3:4
TAXES       RANGE      TAX2:5
MIGRATION-WET RANGE    MIG1:9
SCHOOL-EXP  RANGE      SCH3:3
STATE       REGION     FARM
MINORITY    RANGE      MW1:2
Organizing - IOWA NEBRASKA
```

Figure 4: Initial Generalization

Finally, in Figure 5 we show how UNIMEM was able to

generalize more specific concepts when an attempt was made to store additional instances under the GEN-NODE GND1. (Note that this means that the seven states shown in Figure 3 were not all stored under GND1 at any single time.) For example, Kansas and Nebraska are farm states with low migration rates. This typifies the process in GBM that rarely allows large numbers of instances to end up stored under a given GEN-NODE -- further generalization inevitably creates new concepts which are added to the concept hierarchy in GBM and used to help organize instances.

GND15:
STATE SIZE SIZE:6
MIGRATION-NET RANGE MIG1:9
Organizing: IOWA, MICHIGAN, PENNSYLVANIA

GND0:
STATE REGION FARM
MIGRATION-NET RANGE MIG1:9
Organizing: KANSAS, NEBRASKA

GND25:
STATE SIZE SIZE:6
INDUSTRY TYPE MANUFACTURING
Organizing: MONTANA, TEXAS

Figure 5: Sub-generalizations of GND1

5 RESEARCHER

As mentioned earlier in this paper, RESEARCHER is a program that reads patent abstracts and adds information from them to a Generalization-Based Memory so that it can effectively answer questions [Lebowitz 83b]. In this paper, we look only at the process of taking representations of two objects (or, equivalently, a generalized concept and a concrete object) and extracting a generalized concept. The representations we compare are frame-like and primitive-based, concentrating on the physical relations among the various parts of a complex object. (See [Wasserman and Lebowitz 82] for a complete description of the representation scheme.)

In the disc drive domain, typical concepts the generalization process might identify as being useful would be floppy disc drives or double sided discs. As with all our work, this must be done without specifically providing with examples of these concepts. Instead, instances stored together in Generalization-Based Memory, are recognized as being similar and generalized.

The use of GBM is more complex here than in the UNIMEM. The "features" that two objects have in common can only be determined by comparing two complex object representations. The matching problem is much the same as that faced by Winston in his blocks world learning work [Winston 72]. Our problem is in certain ways both harder and easier than Winston's. It is harder because we are dealing with much more complex representations. It is simplified, however, at least in the long run, by the existence of an entire GBM, rather than a single model. We believe this will simplify the matching process. We look here both at the complexity of matching object descriptions and at how GBM can simplify the process.

The representations for two similar, slightly simplified disc drive patents, used to test the initial version of RESEARCHER's generalization module are shown in Figure 6.

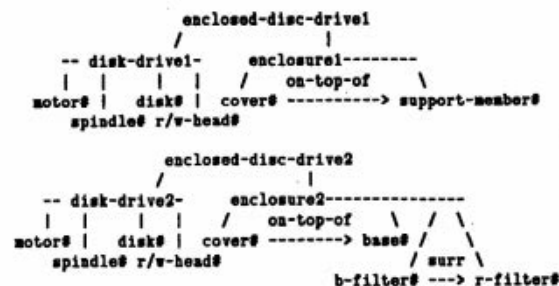


Figure 6: Similar Disc Drives

Clearly the two disc drives in Figure 6 have much in common that can be the source of a new concept derived through generalization -- an "enclosed disc drive". Figure 7 shows the concept created by RESEARCHER's generalization module.

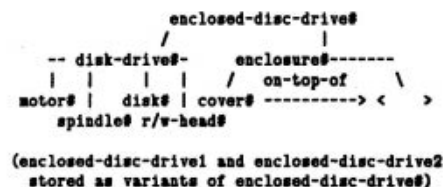


Figure 7: Generalized Enclosed Disc Drive

The idea illustrated in Figure 7 is that RESEARCHER finds the parts of two objects that are similar, and abstracts them out into a generalized concept. In this example, the two devices contained similar disc drives and enclosures. Each had a cover on top of some other object. These similarities form the basis of a generalized enclosed disc drive. Only the additional parts and relations of each instance need be recorded in memory along with the generalization.

Adapting GBM for use on complex structural descriptions has proven to be a difficult problem, even when only considering the assorted relations among the objects in the descriptions. Here we present one of the major problems and suggest the nature of the possible solution.

The central problem in generalizing structural descriptions is the process of matching two representations (either of two objects or an object and a generalized object), thereby determining what parts and relations correspond (as was pointed out for simpler examples in [Winston 72]). Clearly, if we wish to

determine that the disk mounts in two drives are similar, they must be compared with each other. Since one part of the description of complex objects is a set of relations, we must associate the relations in one object with those in the other.

The matching process here is a quite difficult one. Since we are dealing with structured objects, the parts of very similar objects may be aggregated differently in various descriptions. For example, a read/write head might be described as a direct part of a disc drive in one patent, but part of a "read/write assembly" in another. This makes the inherent similarity hard to identify.

At the moment, we deal with this "level problem" with simple heuristics that allow only a limited amount of "level hopping" during the comparison process (to avoid the need to consider every possible correspondence among levels), and a bit of combinatoric force.

We feel that the ultimate solution to the level problem lies in more extensive use of Generalization-Based Memory. If a new object can be identified as an instance of a generalized concept, with only a few minor differences (done with a discrimination-net-based search of the sort described in Section 3), then the levels of aggregation will be set. In effect, by using the form of aggregation that was generalized, we will have a canonical framework for describing new objects. In addition, by using Generalization-Based Memory, we need compare only a small number of differences between objects, rather than complex descriptions. This should allow RESEARCHER to meet all our performance constraints (i.e. be incremental, handle large numbers of objects, and generalize pragmatically), even using the complex representations needed to describe real world objects.

6 CONCLUSION

We believe that our work with Generalization-Based Memory has several important morals. The first is that the development of a dynamic set of concepts is a powerful approach to take when learning from a rich input domain. We can never hope to find the "right" set of concepts, so it is crucial that we constantly update the concepts that we have and look for new ones. Furthermore, the use of long-term memory, in the form of GBM, allows us to deal with many concepts at once, and still retain efficiency. In fact, as we have shown considering many concepts at once often ends up being easier than one at a time and certainly leads to more powerful systems.

REFERENCES

- [Charniak, et al. 80] Charniak E., Riesbeck, C. K., and McDermott, D. V. *Artificial Intelligence Programming*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1980.
- [Collins 78] Collins, A. Fragments of a theory of human plausible reasoning. TINLAP-2, Urbana-Champaign, Illinois, 1978.
- [Kolodner 80] Kolodner, J. L. Retrieval and organizational strategies in conceptual memory: A computer model. Technical Report 187, Yale University Department of Computer Science, 1980.
- [Langley 81] Langley, P. "Data-driven discovery of natural laws." *Cognitive Science* 5, 1 (1981), 31 - 54.
- [Lebowitz 80] Lebowitz, M. Generalization and memory in an integrated understanding system. Technical Report 186, Yale University Department of Computer Science, 1980. PhD Thesis
- [Lebowitz 82] Lebowitz, M. "Correcting erroneous generalizations." *Cognition and Brain Theory* 5, 4 (1982), 367 - 381.
- [Lebowitz 83a] Lebowitz, M. "Generalization from natural language text." *Cognitive Science* 7, 1 (1983), 1 - 40.
- [Lebowitz 83b] Lebowitz, M. Intelligent information systems. Proceedings of the Sixth International ACM SIGIR Conference, ACM SIGIR, Washington, DC, 1983.
- [Lebowitz 83c] Lebowitz, M. Classifying Numeric Information. Columbia University Department of Computer Science, 1983.
- [Michalski 80] Michalski, R. S. "Pattern recognition as rule-guided inductive inference." *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1980).
- [Michalski 83] Michalski, R. S. "A theory of methodology of inductive learning." *Artificial Intelligence* 20 (1983), 111 - 161.
- [Mitchell 78] Mitchell, T. M. Version spaces: An approach to concept learning. Technical Report HPP-79-2, Stanford University, Stanford, California, 1978.
- [Schank 82] Schank, R. C. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, 1982.
- [Wasserman and Lebowitz 82] Wasserman, K. and Lebowitz, M. Representing complex physical objects in memory. Columbia University Department of Computer Science, 1982.
- [Winston 72] Winston, P. H. Learning structural descriptions from examples. In P. H. Winston, Ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, 1972.

The Chunking of Goal Hierarchies: A Generalized Model of Practice

Paul S. Rosenbloom and Allen Newell
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

This work describes recent advances in the specification and implementation of a model of practice. In previous work we showed that there is a ubiquitous regularity underlying human practice — the *power law of practice*. We also developed an abstract model of practice — the *chunking theory of learning*. This previous work established the feasibility of the chunking theory for a single task — a 1023-choice reaction-time task — but the implementation was specific to that one task. In the current work we develop a modified formulation of the chunking theory that allows a more general implementation. In this formulation, task algorithms are expressed in terms of hierarchical goal structures. These algorithms are simulated within a goal-based production-system architecture designed for this purpose. *Chunking* occurs during task performance in terms of the parameters and results of the goals experienced. It improves the performance of the system by gradually reducing the need to decompose goals into their subgoals. This model has been successfully applied to the task employed in the previous work, and to a set of new tasks.¹

Key Terms: Practice, learning, machine learning, artificial intelligence, cognitive science, chunking, goal processing, production systems.

1. Introduction

How can systems — both natural and artificial — improve their own performance? At least for natural systems (people, for example), we know that *practice* is effective. A system is engaged in practice when it repeatedly performs one, or a set of similar, tasks. Recently, Newell and Rosenbloom [1981] brought together the evidence that there is a ubiquitous law — the *power law of practice* — that characterizes the improvements in human performance during practice. The law states that when human performance is measured in terms of the time to perform a task, it improves as a power-law function of the number of times the task has been performed (called the *trial number*). This result holds over the entire domain of human performance, including both purely perceptual tasks, such as target detection [Neisser, Novick, & Lazar, 1963], and purely cognitive tasks, such as

supplying justifications for geometric proofs [Neves & Anderson, 1981] or playing a game of solitaire [Newell & Rosenbloom, 1981].

The ubiquity of the power law of practice suggests that it may reflect something in the underlying *cognitive architecture*. The nature of the architecture is of fundamental importance for both Artificial Intelligence and Psychology (in fact, for all of Cognitive Science). It provides the control structure within which thought occurs; determining which computations are easy and inexpensive, and what errors will be made and when. Take the recent flux of work on expert systems such as MYCIN [Shortliffe, 1976] and R1 [McDermott, 1980] for example. Two important ingredients in such systems stem from fundamental work on the cognitive architecture; specifically, the development of *production systems* [Newell & Simon, 1972; Newell, 1973] and *goal-structured problem solving* [Ernst & Newell, 1969; Newell & Simon, 1972]. This article discusses recent efforts to take advantage of the power law of practice by using it as a generator for a production-system practice mechanism. This is a highly constrained task because of the paucity of plausible practice models that can produce power-law practice curves [Newell & Rosenbloom, 1981].

As a beginning, Newell and Rosenbloom [1981] developed an abstract model of practice based on the concept of *chunking* — a concept already established to be ubiquitous in human performance — and derived from it a practice equation capable of closely mimicking a power law. We hypothesized that this model formed the basis for the performance improvements brought about by practice. In Sections 2 and 3 we briefly summarize this work on the power law and the abstract formulation of the chunking theory.

Rosenbloom and Newell [1982; In press] took the analysis one step further by showing how the *chunking theory of learning* could be implemented within a highly parallel, activation-based production system — *Xaps2* — for a 1023-choice reaction-time task [Seibel, 1963]. This work established more securely that the theory is a viable model of human practice, by showing how it could actually be applied to a task to produce power-law practice curves. By producing a working system, it also established the theory's viability as a practice mechanism for artificial systems.

The principal weakness of the work done up to that point, was that the implementation was heavily task dependent. Both the representation used for describing the task, and the chunking mechanism itself had built into them knowledge about the specific task and how it should be performed. To produce a generalized model of practice, this flaw must be removed by developing generalized, task-independent models of performance and practice.

The first component of the mission — generalizing the model

¹ This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

We would like to thank John Laird for innumerable helpful discussions about this material.

of task performance — is discussed in detail in Rosenbloom [1983]. It is tackled by examining a set of tasks that fall within a neighborhood around the Seibel [1963] task — *stimulus-response compatibility* tasks. The area of compatibility is a subdomain of the set of tasks referred to as *reaction time tasks*. These tasks, which are used widely in psychological experimentation, involve the presentation to a subject of a stimulus display — such as an array of lights, a string of characters on a computer terminal, or a spoken word — for which a specific “correct” response is expected. The response may be spoken, manual — such as pressing a button, pushing a lever, or typing — or something quite different. From the subject’s reaction time — the time it takes to make the response — and error rate, it is possible to draw conclusions about the nature of his cognitive processing.

We will not pursue the analysis of compatibility in detail here, except to note that models based on the concept of *goal hierarchies* provide excellent fits to the data [Rosenbloom, 1983]. In a goal hierarchy, the root node expresses a desire to do a task. At each level further down in the hierarchy, the goals at the level above are decomposed into a set of smaller goals to be achieved. Decomposition continues until the goals at some level can be achieved directly. This is a common control structure for the kinds of complex problem-solving systems found in artificial intelligence, but this is the first time they have been applied to the domain of reaction time tasks.

These goal structures (discussed in Section 4) provide us with the generalized task representation required as the basis for a task-independent chunking mechanism. Chunking essentially implements a form of store-versus-compute trade-off, in which patterns of parameter values are related to patterns of results. This practice mechanism is described in Section 5.

The chunking and goal processing mechanisms were implemented as part of the architecture of the *Xaps3* production system. The *Xaps3* architecture is an evolutionary development from the *Xaps2* architecture. Only those changes necessitated by the needs of chunking and goal processing were made. This architecture is described in Section 6.

In Section 7 we present some of the new results generated from simulations of the practice and compatibility tasks within the new architecture².

In Section 8 we place this work in perspective by relating the chunking theory to previous work on learning mechanisms. The theory stakes out a position intermediary between three previously disparate mechanisms, bringing out an unexpected commonality among them.

In Section 9 we propose a way in which the scope of the chunking theory can be expanded to more than just speeding up existing algorithms. We describe a way in which chunking could be lead to perform such other varieties of learning as *method acquisition* — the way in which a system first develops a performance algorithm for a task.

Finally, in Section 10, we conclude and summarize.

2. The Power Law of Practice

Performance improves with practice. More precisely, the time to perform a task decreases as a power-law function of the number of times the task has been performed. This basic law —

known as the *power law of practice* or the *log-log linear learning law*³ — has been known since Snoddy [1926]. While this law was originally recognized in the domain of motor skills, it has recently become clear that it holds over a much wider range of human tasks — possibly extending to the full range of human performance. Newell and Rosenbloom [1981] showed this for perceptual-motor skills [Snoddy, 1926; Crossman, 1959], perception [Kollers, 1975; Neisser, Novick, & Lazar, 1963], motor behavior [Card, English & Burr, 1978], elementary decisions [Seibel, 1963], memory [Anderson, 1980], routine cognitive skill [Moran, 1980], and problem solving [Neves & Anderson, 1981; Newell & Rosenbloom, 1981].

Practice curves are generated by plotting task performance against trial number. This cannot be done without assuming some specific *measure* of performance. There are many possibilities for such a measure, including such things as quantity produced per unit time and number of errors per trial. The power law of practice is defined in terms of the *time* to perform the task on a trial. It states that the time to perform the task (T) is a power-law function of the trial number (N):

$$T = BN^{-\alpha} \quad (2.1)$$

If Equation (2.1) is transformed by taking the logarithm of both sides, it becomes clear why power-law functions plot as straight lines on log-log paper:

$$\log(T) = \log(B) + (-\alpha) \log(N) \quad (2.2)$$

Figure 2-1 shows the practice curve from one subject in Kollers [1975] study of reading inverted texts — each line of text on the page was turned upside down — as plotted on log-log paper. The solid line represents the power-law fit to this data. Its linearity is clear ($r^2 = 0.932$).

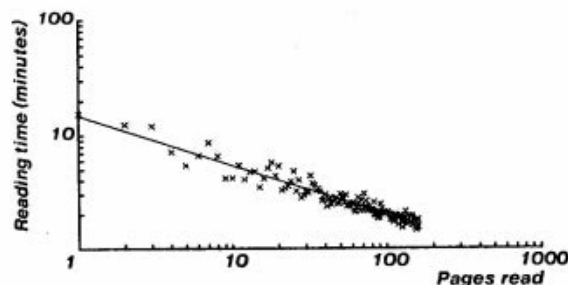


Figure 2-1: Learning to read inverted text (log-log coordinates). Plotted from the original data for Subject HA [Kollers, 1975].

Many practice curves are linear (in log-log coordinates) over much of their range, but show a flattening at their two ends. These deviations can be removed by using a four-parameter *generalized power-law function*. One of the two new parameters (A) takes into account that the asymptote of learning can be greater than zero. In general, there is a non-zero minimum bound on performance time, determined by basic physiological

² A more comprehensive presentation and discussion of these results can be found in Rosenbloom [1983].

³ Power-law curves plot as straight lines on log-log paper.

limitations and/or device limitations — if, for example, the subject must operate a machine. The other added parameter (E) is required because power laws are not translation invariant. Practice occurring before the official beginning of the experiment — even if it consists only of transfer of training from everyday experience — will alter the shape of the curve, unless the effect is explicitly allowed for by the inclusion of this parameter. Augmenting the power-law function by these two parameters yields the following generalized function:

$$T = A + B(N + E)^{-\alpha} \quad (2.3)$$

A generalized power law plots as a straight line on log-log paper once the effects of the asymptote (A) are removed from the time (T), and the effective number of trials prior to the experiment (E) are added to those performed during the experiment (N):

$$\log(T - A) = \log(B) + (-\alpha) \log(N + E) \quad (2.4)$$

Figure 2-2 shows a practice curve from a 1023-choice reaction-time task [Seibel, 1963], as it is fit by a generalized power-law function (each data point represents the mean reaction time over a block of 1023 trials). This is a perceptual-motor task in which the task environment consists of a horizontal stimulus array of ten lights, and a response array of ten buttons in a highly compatible one-one correspondence with the lights, each button right below a light. On each trial, some of the lights are *On*, and some are *Off*. The subject's task is to respond by pressing the buttons corresponding to the lights that are *On*.

This curve, which shows flattening at both ends when plotted as a simple power law, is now linear over the whole range of trials. As stated earlier, similar fits are found across all

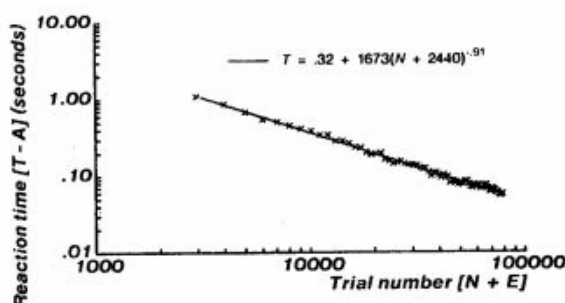


Figure 2-2: Optimal general power law fit to the Seibel data (Log-Log coordinates).

dimensions of human performance. Though these fits are impressive, it must be stressed that the power law of practice is only an *empirical* law. The true underlying law must resemble a power law, but it may have a different analytical form.

3. The Chunking Theory of Learning

The chunking theory of learning proposes that practice improves performance via the acquisition of knowledge about patterns in the task environment. Implicit in this theory is a model of task performance based on this pattern knowledge. These patterns are called *chunks* [Miller, 1956]. The theory thus starts from the *chunking hypothesis*:

- *The Chunking Hypothesis*: A human acquires and organizes knowledge of the environment by forming and storing expressions, called *chunks*, which are structured collections of the chunks existing at the time of learning.

The existence of chunks implies that memory is hierarchically structured as a lattice (tangled hierarchy, acyclic directed graph, etc.), rooted in a set of preexisting *primitives*. A given chunk can be accessed in a top-down fashion, by *decoding* a chunk of which it is a part, or in a bottom-up fashion, by *encoding* from the parts of the chunk. Encoding is a recognition or parsing process.

The existence of chunks does not need to be justified solely on the basis of the practice curves. Chunks stand on their own as a thoroughly documented component of human performance [Miller, 1956; DeGroot, 1965; Bower & Winzenz, 1969; Johnson, 1972; Chase & Simon, 1973; Chase & Ericsson, 1981]. The traditional view of chunks is that they are data structures representing the combination of several items. For example, in one set of classic experiments, Bower and colleagues [Bower, 1972; Bower & Springston, 1970; Bower & Winzenz, 1969] showed that recall of strings of numbers or letters is strongly influenced by the segmentation of the string. If the segmentation corresponds to a previously learned grouping of the items (for example, FBI-PHD-TWA-IBM), performance is better than if it is contradictory (FB-IPH-DTW-AIB-M). These results were interpreted as evidence for segmentation guided chunking of familiar strings. By replacing a string of several letters with a single chunk, the subject's memory load is reduced, allowing more letters to be remembered. At recall time the chunks are decoded to yield the original items to be recalled.

The chunking theory of learning proposes two modifications to this classical view. The first change is to assume that there is not a single symbol (chunk) to which the items are encoded, and from which they can later be decoded. As a simple example, the process of reading the string IBM out loud is more than just the encoding of the three letters into a chunk, followed by the subsequent decoding to the three letters. What needs to be decoded is not the visual representation of IBM, but the articulatory representation — allowing the subject to say IBM.

Based on this consideration, the chunking theory assumes that there are two symbols for each chunk — a *stimulus* symbol and a *response* symbol. The process of using a chunk consists of encoding the stimulus items to the stimulus symbol (a many-one mapping), mapping the stimulus symbol to the response symbol (a one-one mapping), and decoding the response symbol to the response items (a one-many mapping) (Figure 3-1). Encoding and decoding are fast hierarchical processes, while the mapping serves as a (serial) point of control at which the choice of response can be made. Acquisition of a chunk speeds up performance by reducing the number of mappings to be performed. In the example in Figure 3-1, before the chunk is acquired, three mappings (one for each letter) are required. After acquisition of the chunk, one mapping suffices.

The second difference between this proposal and the classical view of chunking is that the chunk consists of the three processes (encoding, mapping, and decoding), not just the symbols. This is really just a shift of emphasis, in which chunks are viewed as the processes, rather than just as the results of the processes.

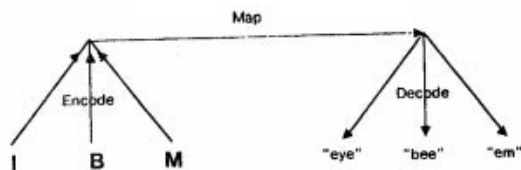


Figure 3-1: A three-part chunk for the articulation of the visual string "IBM".

4. Goal-Structured Performance Models

In its current formulation, chunking explains how performance on a task can be sped up with practice. It does not explain how the organism first learns to do the task (but see the discussion in Section 9). Consequently, each task simulation must be initialized with a performance model for the task. What is needed — and is provided by the notion of a *goal hierarchy* — is a general, task-independent representation for these performance models. Goal hierarchies are found frequently in artificial intelligence systems, but have not previously been employed in the modeling of the kinds of reaction-time tasks with which we are dealing.

Goal hierarchies are built out of *goals*, where each goal is a data structure representing a desired state of affairs. A goal is not a procedure for bringing about that state; it is only a description of the state. In order to bring about the goal state, there must be a *method* associated with the goal. The method could be a rigid algorithm, or it could be one of the more flexible *weak methods* [Newell, 1969] such as means-ends analysis [Ernst & Newell, 1969] or heuristic search [Nilsson, 1971]. Goals take *parameters* — information used in the fulfillment of the goal — and return *results*.

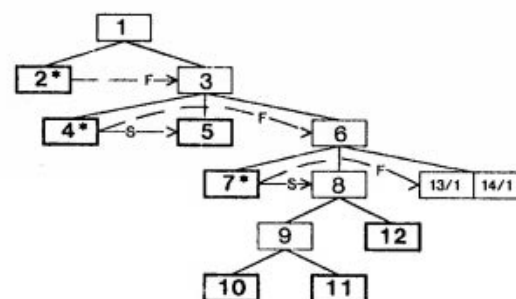
When a goal can be decomposed into a set of simpler goals [Nilsson, 1971], and those goals can be decomposed even further, a goal hierarchy results. In its simplest form, as an AND hierarchy, a goal is successful if all of its subgoals are successful. The structure to be described here, more closely resembles an AND/OR hierarchy, in which some goals succeed only if all of their subgoals succeed, and others succeed if any one of their subgoals succeed. A *terminal* goal is reached when the goal can be fulfilled directly, without the need for further decomposition.

One strategy for processing a goal hierarchy is to attempt the goals in a *depth-first* fashion, in which the most recently generated (the deepest) goal is always selected as the next goal to process. With a depth-first paradigm, there is always exactly one goal being actively worked on at any point in time. We will refer to this goal as the *active* or *current* goal. When a subgoal becomes the active goal, the parent goal of that subgoal is *suspended* until control is returned to it by completion of the subgoal, at which point it again becomes the current goal. On completion, the subgoal will either have *succeeded* or *failed*. The *control stack* specifies the location in the hierarchy at which control currently resides. It does this by maintaining the path from the root goal of the hierarchy to the current goal. This path consists of the active goal, and all of its suspended ancestors.

Figure 4-1 shows two different representations of a goal hierarchy for the Seibel [1963] task. At the top of the figure, the

hierarchy is shown as a tree structure. At the bottom of the figure, the goals are shown in their depth-first processing order. In both representations, the bold-faced goals are the terminals — those goals that can be fulfilled directly. The labeled arrows (either S or F) represent *branch* points in the hierarchy. With simple depth-first processing, the children of a node are processed in a strict left-to-right fashion. We have augmented this style of control by allowing the left-to-right processing to be conditioned on the success or failure of the previous child (actually of any previous goal for which this information is still available). For example, goal number 2 (No-Light-On?) tests whether there are any lights on within a specific region of the display of lights. Only if this goal fails, should goal 3 (Do-Lights) be attempted. If goal 2 succeeds, then there are no lights on that need to be processed, and the parent goal (goal 1: Do-Lights-If-Any) can just be terminated successfully. Those goals that do not return a result are called *predicates* (denoted by a * in Figure 4-1), and are used to test the truth of various conditions. It is only the success or failure of predicates that matters (as the basis for a branch). If the predicate succeeds, then the condition is true. If it fails, it could be either that the condition is false, or that something went wrong during the attempt.

The goal structure in Figure 4-1 is based on a recursive divide-and-conquer algorithm in which the stimulus display is broken up into smaller and smaller horizontal segments, until *manageable* pieces are generated. There are three types of horizontal segments that have been defined as manageable. The first type of manageable segment is one in which no lights are on. Such segments require no explicit processing, so the goal just returns with success. The opposite of the first type of segment — one in which no lights are off — is also manageable. For such a segment, the system generates a single response



1. Do-Lights-If-Any(Min-X, Max-X)
2. No-Light-On?(Min-X, Max-X)
- IF FAILED No-Light-On? THEN
3. Do-Lights(Min-X, Max-X)
4. No-Light-Off?(Min-X, Max-X)
- IF SUCCEEDED No-Light-Off? THEN
5. Do-Press-All-Buttons(Min-X, Max-X)
- IF FAILED No-Light-Off? THEN
6. Do-Off-And-On(Min-X, Max-X)
7. One-Light-On?(Min-X, Max-X)
- IF SUCCEEDED One-Light-On? THEN
8. Press-Button-Under-On-Light(Min-X, Max-X)
9. Get-On-Light-X(Min-X, Max-X)
10. Get-On-Light-Stimulus(Min-X, Max-X)
11. Get-Stimulus-X
12. Press-Button-At-X
- IF FAILED One-Light-On? THEN
13. Do-Lights-If-Any(Min-X, [Min-X + Max-X]/2)
14. Do-Lights-If-Any([Min-X + Max-X]/2, Max-X)

Figure 4-1: Goal hierarchy for the Seibel [1963] task.

specifying that a *press* action must occur in the entire region

defined by the segment (using the **Do-Press-All-Buttons** goal). Specifying a single button press is actually a special case of this, in which the region is just large enough to contain the one button. Allowing multi-on-light segments to be manageable, implies that sequences of adjacent on-lights can be pressed simultaneously, even before chunking has begun. Such behavior is seen very early in the trial sequence for some subjects. The remaining manageable segments are those that contain exactly one light on. These segments are processed (using the **Press-Button-Under-On-Light** goal) by finding the location of that light, and generating a button press at that location. If a generated segment does not meet any of these three criteria, it is unmanageable, and is split into two smaller segments.

The recursive aspect of the algorithm implies that many different instances of each goal will be simultaneously represented in the system, though at most one can actually be active. In order to keep track of which goal instance is relevant to which segment of the stimulus display, the segment (in terms of its minimum and maximum X values) is an *explicit* parameter to the goals. In addition to the explicit parameters, a goal can have *implicitly* defined parameters. Any object existing before the activation of the goal (i.e., part of the goal's *initial state*), that is examined during the processing of the goal — such as a stimulus light — is an implicit parameter of the goal. Implicit parameters are dynamically determined by the actual processing of the goal.

A second implication of the recursion in this hierarchy, is that the system does not start off with a constant built-in goal hierarchy. Instead, it has a generator of goal hierarchies. That chunking works on such a structure, is important for any claims about the potential generality of the mechanism (for more on this, see Section 9).

The recursion occurs at goals 13 and 14 in Figure 4-1. They are repetitions of the topmost goal in the hierarchy (**Do-Lights-If-Any**), but the scope of each is limited to one half of the display currently being processed. The numeric computation to obtain the middle of the segment (involving an addition and a division), while on the surface too powerful a computation to appear where it does, is only intended as an approximation to a process that divides the stimulus display into two (or three) roughly equal parts.

In addition to the goal hierarchy, the model assumes a *working memory* for the short-term storage of information relevant to the processing that is going on. For each goal, the working memory is logically partitioned into two components — the *initial state* and the *local state*. The initial state consists of the data existing at the time the goal is first activated. The remainder of the working memory — consisting of those pieces of data created during the processing of the goal — make up its local state. Only the local state of a goal can be modified during the processing of that goal; the initial state can be examined, but not modified. The modularity resulting from this scoping rule increases the likelihood that an arbitrary set of goals can be pursued without interfering with each other. This modularity is also important in ensuring correct performance of the chunking mechanism.

Each datum in the working memory is relevant in a particular temporal *processing context*. As long as the datum's creator goal is in the control stack, the system is either working on that goal, or on one of its descendants. The datum may be relevant at any point in this processing. However, once the goal disappears from the control stack, it is no longer being pursued.

The datum will most likely no longer be relevant. The processing context of a datum is that period of processing during which its creator goal is in the control stack. Once a piece of information becomes *out of context* — its creator is no longer part of the control stack — it usually can be safely deleted from the working memory. Data that cannot be deleted — because they are needed outside of the context in which they were created — are called the *results* of the goal. Their continued presence in the working memory is assured by changing their context to be the context of the parent goal.

5. Chunking on Goal Hierarchies

Given a task-independent organization for performance models, we can return to the original objective to describe a task-independent formulation of the chunking theory. In the chunking mechanism described by Rosenbloom and Newell [1982; In press], chunks related patterns of stimuli (lights) to patterns of responses (button presses). The goal-oriented formulation is obtained by altering this definition slightly so that chunks relate patterns of goal parameters to patterns of goal results. Each chunk improves the performance of the system by eliminating the need to fully process a specific instance (a combination of parameter values) of a particular goal. It replaces the normal processing (decomposition into subgoals for non-terminal goals, and direct execution of an action for terminal goals) with a direct connection between the relevant parameter values and results. A goal can (and almost certainly will) have more than one possible chunk; each combination of parameter values requires its own chunk. It should be clear that chunking implements a store-versus-compute trade-off, a form of goal-based caching.

As with the abstract characterization of the chunking theory, each chunk consists of three components: *encoding*, *decoding*, and *connection* (or *mapping*). The goal's parameter values form the basis for the encoding component. Given the presence of those values in the working memory, the encoding component generates a new object representing their combination. Encoding is a parallel, goal-independent, data-driven process. Every encoding component executes as soon as appropriate, irrespective of whatever else is happening in the system. The results of encoding components can themselves become parameter values to other goals, leading to a hierarchical encoding process.

The results of the goal form the basis for the decoding component. Given the presence of an encoded result-object in the working memory, the decoding component generates the actual results returned by the goal. Decoding occurs when the results are needed. As with encoding, the set of decoding components forms a hierarchical structure, in which complex results are decoded to simpler ones, which are then decoded even further.

The connection component of the chunk generates the encoded result from the encoded parameter. Connections provide a locus of control by occurring serially, under the control of the goals. Thus, a connection can be made only when the system is working on the goal for which the chunk was formed (and after the encoding component has executed). This assures that, even though encoding and decoding are uncontrolled, only appropriate results are generated.

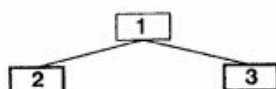
Chunks are created bottom up in the goal hierarchy. A chunk can be created for a goal when the following two conditions are met: (1) the goal has just completed successfully, and (2) all of

the goal's subgoals were themselves processed by chunks. It is this bottom-up aspect of chunking that leads to hierarchical encoding and decoding networks. However, notice that bottom-up chunking does not imply that all low-level chunks are learned before any high-level chunks are learned, or even that all of the chunks must be learned for a subgoal before any can be learned for its parent goal. The second condition on chunk creation merely states that chunks must exist for the goal's subgoals *in the current situation*. Whether other chunks exist or do not exist for the subgoals is irrelevant.

Given enough practice, all of the situations for all of the goals in the hierarchy will be chunked, and asymptotic behavior will be reached for the task. The amount of time this takes depends on the number of goals, the number of situations for each goal, how frequently the different situations arise, and whether chunks are created whenever they can be.

As a simple example of how chunking works, consider the simple three-goal hierarchy in Figure 5-1. This structure computes the average of two numbers. The top-level goal (Compute-Average-Of-Two-Numbers) takes as parameters the two numbers to be averaged, and returns a single result, which is their mean. The first subgoal (Compute-Sum-Of-Two-Numbers) performs the first half of the computation. It takes the two numbers as parameters, and returns their sum as its result. The second subgoal finishes the computation by taking the sum as a parameter, and returning half of it as its result.

Suppose that the first task is to average the numbers 3 and 7. Control would pass from goal number 1 to goal number 2. When goal 2 finishes and returns its result of 10, a chunk of three components is created (bottom left of Figure 5-2). An encoding



1. Compute-Average-Of-Two-Numbers
2. Compute-Sum-Of-Two-Numbers
3. Divide-Sum-By-2

Figure 5-1: A simple three-goal hierarchy for the averaging of two numbers.

component is created that encodes the two parameters (3 and 7), into a new symbol (E1). It executes as soon as it is created, because the parameters are in the working memory. A decoding component is created that, when necessary, decodes from a second new symbol (D1) to the result (10). A connection component (the horizontal line with the goal name above it and goal number below it) is created that generates the result symbol (D1) when it detects both the presence of the encoded parameter (E1) and that goal 2 is the active goal. The connection does not execute immediately, because goal 2 is already complete when the chunk is created.

Following the termination of goal 2, goal 1 is reactivated, but then is suspended in favor of goal 3 (Divide-Sum-By-2). When this goal terminates successfully (returning the number 5), a chunk is created for it (bottom right of Figure 5-2). The encoding component encodes the number 10 into the symbol E2; the decoding component decodes from the symbol D2 to the

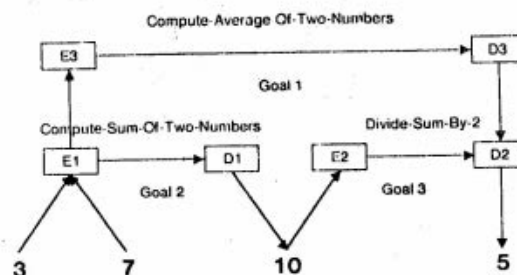


Figure 5-2: Sample chunks created for the hierarchy in Figure 5-1.

number 5; and the connection component connects E2 to D2 (in the presence of an active goal 3). In contrast to the chunk for goal 1, this chunk can be used in more than one task situation. It can be used whenever goal 1 generates a sum of 10, whether it does it by adding 3 and 7, 5 and 5, or any other pair of numbers. This is a form of transfer of training.

Following the termination of goal 3, goal 1 is reactivated, and terminates successfully (returning the number 5). No chunk is created for goal 1 because its subgoals were not processed by chunks. At this point, the task is complete.

Given what was learned during the performance of this task, the next time the same task is performed things will go differently. As soon as the task is restarted (again with the values 3 and 7), the encoding component from the chunk for goal 2 executes, placing E1 in the working memory. Goal 1 is activated, and then suspended in favor of goal 2. At this point, the connection component for goal 2 executes, generating D1 and successfully completing goal 2. D1 is decoded to the number 10, which is then immediately reencoded to E2 by the encoding component for goal 3. Following the subsequent reactivation and suspension of Goal 1, Goal 3 is activated. The connection component for Goal 3 executes, generating D2, and returning D2 as the result to goal 1. This time, when goal 1 terminates, a chunk is created (top of Figure 5-2), because both of the subgoals were processed by chunks.

The encoding component for this chunk builds upon the existing encodings by encoding E1 to a new symbol (E3); it does not go straight from goal 1's primitive parameters (3 and 7). This happens (and causes hierarchical encoding) because, for this instance of goal 1, E1 is the implicit parameter, not 3 and 7. Recall from Section 4 that the implicit parameters of a goal consist of those pieces of the goal's initial state that are examined during the goal's performance. E1 is generated before goal 1 is activated (so it is part of the goal's initial state), and examined by the connection component for goal 2. On the other hand, neither of the objects representing the numbers 3 and 7 are examined during the processing of goal 1. Therefore, E1 is an implicit parameter (and included in the chunk), while the numbers 3 and 7 are not.

The decoding component is created in a similarly hierarchical fashion. It decodes from a new symbol (D3) to D2. This occurs because D2 (and not the number 10) is the result of goal 1. It never became necessary to decode D2, so it was passed directly up as the result of both goals 3 and 1. The connection component of this chunk links E3 to D3 in a straightforward manner.

If the same task is performed yet again, the encoding components immediately generate E1, followed by E3. Goal 1 is activated, and its connection component executes, generating D3 and completing goal 1. If the result is needed by some part of the system outside of the hierarchy, it will be decoded to D1, and then to the number 10.

The example that we have just gone through outlines the basics of how the chunking mechanism works. The next step is to look at the more complex situation of the Seibel task (Figure 4-1). Chunking starts in this structure with the terminal goals (numbers 2, 4, 5, 7, 10, 11, and 12). Take goal 11 (Get-Stimulus-X), for example. Successful completion of this goal requires retrieving from the working memory the representation of a stimulus that has been perceived, and then generating a piece of information representing the horizontal location (X) of that stimulus. The parameter for this goal is the stimulus — an implicit parameter — and the result is the location. In the chunk for this situation, the encoding and decoding components are trivial. They simply recode from the single parameter to a new symbol, and from a second new symbol to the result. The connection component tests for the presence of an active Get-Stimulus-X goal and the encoding symbol, and produces the decoding symbol.

Goal 10 (Get-On-Light-Stimulus) presents a slightly more complicated case than goal 11. The goal has both explicit and implicit parameters. The explicit parameters (Min-X and Max-X) define the region of the stimulus display in which an on-light should be found. The implicit parameter is the actual external representation (in the stimulus display) of the on-light that is found. The result of this goal is the internal representation of that on-light, as it appears in the working memory. Just as before, the implicit parameter and the result form the basis for trivial encoding and decoding components. However, the explicit parameters are different. They do not exist as separate entities in the working memory; instead they act as if they were augmentations to the name of the goal. Therefore, they appear along with the name of the goal, as part of the connection component.

As a final example of the workings of the chunking mechanism, consider how it creates the kinds of multiple light-button chunks produced by the chunking mechanism in Rosenbloom and Newell [In press]. The locus of these chunks can be found at the recursive step in the goal hierarchy. The root (and recursive) goal in the hierarchy (Do-Lights-If-Any) has implicit parameters that represent lights in the stimulus display, and generates results that are the button presses for those lights. The earliest chunks that can be created for this goal are those at the bottom of the recursion; that is, goals to process manageable segments of the display. Each of these chunks will represent a single on-light in a region, a region of solid on-lights, or a region with no on-lights. Once the chunks exist for goals 13 and 14 (and their sibling goal 7: the predicate One-Light-On?) in a single situation, the parent goal (goal 6: Do-Off-And-On) can be chunked. This yields a new chunk for the combined situation in both segments. This process continues up the hierarchy until goal 1 is chunked for that level of recursion. But goal 1 at that level is just goal 13 or 14 at the next level up. Therefore, gradually (since these chunks are acquired one at a time), the level of aggregation of segments covered by chunks increases.

This process always leads to light-button chunks for contiguous light-button pairs. It does not lead to chunks for disjoint patterns such as just the two extreme right and left light-

button pairs. This is not a limitation on the generality of the chunking mechanism. Instead, it is a function of the goal structure employed. A different goal structure (reflecting a different processing strategy) could lead to the creation of such disjoint chunk patterns.

One of the strong task-dependencies present in the Rosenbloom and Newell [In press] chunking mechanism was that encoding productions had to have a condition added to them that assured that no on-light appeared between the two patterns being chunked together. This was so, even though the information that this condition was necessary appeared nowhere in the task algorithm. In the hierarchy in Figure 4-1, such conditions are generated naturally from the goals for processing segments of the display with no on-lights in them. We see that the task dependencies show up in the goal hierarchy, not in the chunking mechanism.

The following list of points summarizes the key aspects of chunking as it applies to goal hierarchies.

- Each chunk represents a specific goal with a specific set of parameter values. It relates the parameter values to the results of the goal.
- Chunks are created through experience with the goals processed.
- Chunks are created bottom up in the goal hierarchy.
- Chunks consist of connection, encoding, and decoding components.
- Chunk encoding and decoding are hierarchical, parallel, goal-asynchronous processes that operate on goal parameters and results (respectively).
- Chunk connection is a serial, goal-synchronous process that generates (possibly encoded) results from (possibly encoded) parameters.
- Chunks improve performance by replacing the normal processing of a goal (and its subgoals) with the faster processes of encoding, connection, and decoding.

6. The Xaps3 Architecture

The *Xaps3* production-system architecture was designed to be used in the modeling of goal hierarchies and chunking. It is a new architecture, but it builds upon the work done in the development of the *Xaps2* architecture. In their presentation of *Xaps2*, Rosenbloom and Newell [In press] began by discussing a set of *constraints* that must be met by any architecture within which the chunking theory of learning is implemented. We don't claim that these constraints are known to be necessary — the arguments are not that tight. But this permits (and encourages) attempts to show how the constraints can be circumvented.

The same constraints still hold — as they must if they are really constraints — for the design of the *Xaps3* architecture. More recently, two new constraints have been formulated from the consequences of the chunking theory. These new constraints rule out *Xaps2*, and have led to the design of the *Xaps3* architecture. Though it is a direct descendent of *Xaps2*, the constraints have forced a number of significant changes in the *Xaps3* design.

6.3.1 Recognition

The match phase is quite simple. All of the productions in the system are matched against working memory in parallel (conceptually), to yield a set of legal instantiations — at most one per production. Each instantiation consists of a production name and a set of variable bindings for that production that yield a successful match. This set of instantiations is then passed in its entirety to the conflict resolution phase⁹, where they are winnowed down to the set to be executed on the current cycle. This winnowing is accomplished via a pair of conflict-resolution rules.

The first rule is *goal-context refraction*. It states that a production instantiation can fire only once within any particular goal context. It is a form of the standard *Ops* refractory inhibition rule [Forgy, 1979], differing only in how the inhibition on firing is released. With the standard rule, the inhibition is released whenever one of the working-memory objects on which the instantiation is predicated, has been modified. With goal-context refraction, inhibition is released whenever the system leaves the context in which the instantiation fired. If the instantiation could not legally fire before the context was established, and it could fire both during and after the context was established, then the instantiation must be based, at least in part, on a result generated during the context and returned when the context was left. Therefore, the instantiation should be free to fire again to reestablish the still relevant information.

The second rule — the *parameter-passing bottleneck* — states that only one parameter-passing instantiation can execute on a cycle (conceptually selected arbitrarily). This rule first appeared in a slightly different form as an assumption in the *Hpsa77* architecture [Newell, 1980a]. It will be justified in Section 6.5.

6.3.2 Action

All of the instantiations that make it through the conflict resolution phase are fired (conceptually) in parallel. Firing a production instantiation consists of: (1) overwriting the *examined-by* tags of the working-memory objects matched by the instantiation, with the identifier of the active goal; (2) replacing all variables in the actions by the values determined during the match phase; (3) evaluating any *Lisp* forms; and (4) performing the actions. This can result in the addition of new objects to working memory, and the modification of existing local objects. If conflicting values are simultaneously asserted, the winner is selected arbitrarily. This does not violate the crypto-information constraint because the architecture is using no information in making the decision. If the performance system works correctly, even when it can't depend on the outcome of the decision, then the learned information will not lead to incorrect performance.

6.4 Goal Processing

In Section 4 we described the processing of goals at an abstract level. In this section, we make that processing concrete, by describing how it is implemented within the *Xaps3* architecture. The goals themselves, unlike chunks, are just data structures in working memory. A typical *Xaps3* goal goes through four phases in its life. The current phase of a goal is represented explicitly at all times by the value associated with

the *STATUS* attribute of the working-memory object representing the goal.

In the first stage of its life, the goal is *desired* — at some point, but not necessarily right then, the goal should be processed. Productions create goal desires by generating a new object of type *Goal*. Each new goal object must have a *NAME* attribute, a *STATUS* attribute with a value of *Want*, and a *RESULT-TYPE* attribute. In addition, it may have any number of other attributes, specifying explicit parameters to the goal. The value of the *RESULT-TYPE* attribute specifies the type of the results that are to be returned on successful completion of the goal. All local objects of that type are considered to be results of the goal. Results are marked explicitly so that they won't be flushed when the context is left, and so that the chunking mechanism will know to include them in the chunks for the goal.

Leaving the expression of goal desires under the control of productions allows goals to be processed by the architecture, while still leaving the structure of the hierarchy under program (that is, production) control. The architecture controls the transition to the second, *active* phase of the goal's life. At most one goal is active at any point in time: The architecture attempts to activate a new goal whenever the system is at a loss as to how to continue with the current goal. This occurs when there is an empty conflict set; that is, there is no production instantiation that can legally fire on the current cycle. When this happens, the system looks in working memory to determine if there are any subgoals of the current goal — those goals created while the current goal was active — that are desired. If such a subgoal is found, it is made the active goal, and the parent goal is *suspended* by replacing its *STATUS* with the identifier of the newly activated subgoal. If more than one desired subgoal is found, one is arbitrarily selected (actually, the last one found is used).

Suspension is the third phase in the life of a goal (it occurs only for non-terminal goals). Replacing the *STATUS* of the parent goal with the subgoal's identifier accomplishes two things: it halts work on the goal, because the productions that process goals all check for a *STATUS* of *Active*; and it maintains the control stack for the goal hierarchy. A suspended goal remains suspended until its active subgoal terminates, at which time it returns to being active. If a goal has more than one subgoal, it oscillates between the active and suspended states.

If no progress can be made on the active goal, and there are no desired subgoals, then the system has no idea how to continue making progress, so it terminates the active goal with a *STATUS* of *Failed*. Following termination, the goal is in its fourth and final phase of life. In addition to a failure termination, goals can be terminated with a *STATUS* of *Succeeded*. There are no uniform criteria for determining when an arbitrary goal has completed successfully, so it has been left to productions to assert that this has happened. This is done via the creation of an object of type *Succeeded*. When the architecture detects the presence of such an object in working memory, it terminates the current goal, and reactivates its parent.

At goal termination time, a number of activities occur, in addition to the obvious one of changing the active goal. The first two activities occur only on the successful termination of a goal. As discussed in the following section, the first step is the possible creation of a chunk. The second step is to return the results of the terminating goal to its parent goal. This is accomplished by altering the *created-by* tags of the results so that it looks as if they were created by the parent goal. The third step is to delete all of the objects from working memory created during the goal to be terminated. The fourth, and final, step is to enable result decoding if it is appropriate.

⁹For purposes of efficiency, these two phases are actually intermingled. However, this does not change the behavior of the system at the level at which we are interested.

(RESULT-TYPE) specifies the type of object to be returned as the result of this goal (see Section 6.4 for the details).

During the lifetime of a working memory object, two types of auxiliary tags are kept. The first one marks the identifier of the goal that was active — there being at most one — when the object was created (If there was no goal active at the time, then the special symbol <None> is used). This tag is used for determining the object's status as being either part of the local or initial state of the current goal. The second tag marks the identifier of the goal active when the object was last examined by a production. This tag, in combination with the first one, allows the chunking mechanism to determine the implicit parameters of goals.

6.2 Production Memory

There is a single homogeneous production memory in *Xaps3*. The productions in it are similar to the productions in *Xaps2* and to those in the *Ops* languages. They have three parts: a name, a list of one or more conditions, and a list of one or more actions. Here is an example production taken from the implementation of the Seibel task (Figure 4-1)⁷.

```
(DefProd SubGoal/Do-Lights/Do-Press-All-Buttons
  ((Goal <Exists> [NAME Do-Lights] [STATUS Active]
    [MINIMUM-LOCATION = Min-Loc]
    [MAXIMUM-LOCATION = Max-Loc])
   (Goal {<Local> <Exists>} [NAME No-Light-Off?]
    [STATUS Succeeded]))
→
((Goal <New-Object> [Name Do-Press-All-Buttons]
  [STATUS Warn] [RESULT-TYPE Response]
  [MINIMUM-LOCATION = Min-Loc]
  [MAXIMUM-LOCATION = Max-Loc])))
```

The name is purely for the convenience of the programmer, it doesn't affect the processing of the system in any way. Each condition is a pattern to be matched against the objects in working memory. A condition pattern contains a *type* field (specifying the type of the object to be matched), an *identifier* field (containing several different types of information), and an optional set of patterns for attributes. Condition patterns are primarily built from *constants* and *variables*. Constants are signified by the presence of the appropriate symbol in the pattern, and only match objects that contain that symbol in that role. Some example constants in production *SubGoal/Do-Lights/Do-Press-All-Buttons* are *Goal*, *MINIMUM-LOCATION*, and *Succeeded*. Variables are signified by a symbol (the name of the variable) preceded by an equal sign. They match anything appearing in their role in an object. An example is *= Min-Loc* in the sample production. All instances of the same variable within a production must be bound to the same value in order for the match to succeed.

The identifiers in production *SubGoal/Do-Lights/Do-Press-All-Buttons* are specified by the special symbol <Exists>. Such a condition succeeds if

there is any object in working memory that matches the condition. If there is more than one such object, the choice is conceptually arbitrary (actually, the first object found is used). Only one instantiation is generated. If the identifier is specified as a variable, the condition still acts like an exists condition, but the identifier can be retrieved, allowing the object to be modified by the actions of the production.

The complete opposite of an exists condition, is a not-exists condition — commonly called a negated condition. When the symbol <Not-Exists> appears in the identifier field, it signifies that the match succeeds only if there is no object in working memory that matches the remainder of the condition pattern.

The second condition in the sample production contains the symbol <Local>, as well as <Exists>, in the identifier field (they are conjoined syntactically by curly-brackets). When this symbol is added to an identifier pattern of any type — constant, variable, exists, and not-exists — it signifies that the condition should only be matched against those objects in working memory that are local to the active goal. This information — provided by the objects' *created-by* tags — is a means by which the productions that work on a goal can determine which objects are part of their local context.

The remainder of the condition specifies a pattern that must be met by the attribute-value pairs of working-memory objects. These patterns are built out of constants, variables, built-in predicates (such as <Greater-Than>), and general *Lisp* computations (via an escape mechanism). In addition, any of the above forms can be negated, denoting that the condition only matches objects that do not match the pattern.

There is only one kind of action in *Xaps3* — modify working memory. As in *Xaps2*, the interface to the outside world is through working memory, rather than through production actions. Actions can create new objects in working memory and, under certain circumstances, modify existing objects. A new object is created when the action has an identifier of <New-Object>, by replacing the identifier with a newly generated symbol, instantiating the variables with their values computed during the match, and replacing calls to *Lisp* functions with their values (via another escape mechanism). An existing object can be modified by passing its identifier as a parameter from a condition. As Discussed in Section 4, only objects local to the current goal can be modified.

There are no production actions that lead to the deletion of values or objects from working memory. A value can be removed only by superceding it by another value. As discussed in Section 4, objects go away when they are no longer part of the current context⁸. No explicit mechanism for deletion has proved necessary, so none has been included.

6.3 The Cycle of Execution

Xaps3 follows the traditional *recognize-act* cycle of production-system architectures, with a few twists thrown in by the need to process goals. The recognition phase begins with the *match*, and finishes up with *conflict resolution*. The *cycle number* — the simulated time — is incremented after the recognition phase, whether any productions are executed or not. During the act phase, productions are executed.

⁷ As with the format of working-memory objects, the format of this production has been slightly altered to increase the clarity of presentation.

⁸ This mechanism bears a relationship to the *dampening* mechanism in the *Act* architecture [Anderson, 1976].

pairs on the goal object. Given the goal's identifier, its implicit parameters are found by retrieving all of the objects in working memory that were part of the goal's initial state — that is, their *created-by* tag contains an identifier different from that of the active goal — and were examined by a production during the processing of the active goal. This last type of information is contained in the objects' *examined-by* tags. The goal's results are equally easy to find. The system simply retrieves all of the goal's local objects that have a type equal to the goal's *RESULT-TYPE*. Because the parameter and result information is determined from the constant objects in working memory, chunks themselves are not parameterized. Each chunk represents a specific situation for a specific goal.

The one complication to the clean picture of chunk acquisition so far presented, concerns the use of negated conditions during the processing of a goal. When a negated condition successfully matches working memory, there is no working-memory object that can be marked as having been examined. Therefore, some of the information required for chunk creation cannot be represented in working-memory. The current solution for this problem is not elegant, but it works. A temporary auxiliary memory is maintained, into which is placed each non-local negated condition occurring on productions that fire during the processing of the goal (local negated conditions can be ignored, because they do not test the initial state). This memory is reinitialized whenever the goal that is eligible to be chunked changes. Before the conditions are placed in this memory, they are fully instantiated with the values bound to their variables by the other conditions in their production.

7. Results

In this section we present some results from applying this architecture to the Seibel and compatibility tasks. A more complete presentation of these results can be found in Rosenbloom [1993]. For the Seibel task, two different sequences of trials were simulated. The first sequence is the same as the one used in Rosenbloom and Newell [In press]. The simulation completed 268 trials before it was terminated¹³. A total of 682 productions were learned. On the second sequence of trials — from a newly generated random permutation of the 1023 possibilities — 259 trials were completed before termination. For this sequence, 652 productions were learned.

Figure 7-1 shows the first sequence as fit by a general power law. Each point in the figure represents the mean value over five data points (except for the last one, which only includes three)¹⁴. For this curve, the asymptote parameter (*A*) has no effect. Only *E*, the correction for previous practice, is required to straighten out the curve. At first glance, it is nonsensical to talk about previous practice for such a simulation, but a plausible interpretation is possible. In fact, there are two independent explanations — either or both may be responsible.

The first possibility is that the level at which the terminal goals are defined is too high (complex). If the "true" terminals are

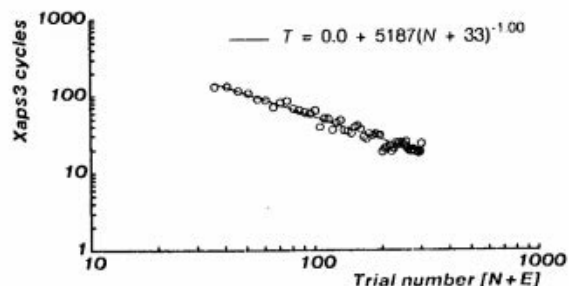


Figure 7-1: General power law fit to 268 simulated trials of the Seibel [1963] task.

more primitive, then chunking starts at a lower level in the hierarchy. One view of what chunks are doing is turning their associated (possibly non-terminal) goals into terminal goals for particular parameter situations. During preliminary bottom-up chunking, the system would eventually reach the lowest level in the current hierarchy. All of the practice prior to that point is effectively previous practice for the current simulation.

The other source of previous practice, is the goal hierarchy itself. At the beginning of the simulation, this structure is already known perfectly. However, there must be a process of *method acquisition* by which the subject goes from the written (or oral) instructions to an internal goal hierarchy. Though method acquisition does not fall within the domain of what we have here defined as "practice", we will propose in Section 9 a scheme whereby chunking may lead to a mechanism for method acquisition.

One problem that cropped up in the simulations of the earlier model of chunking [Rosenbloom & Newell, In press], was that learning was demonstrably too fast. In order to achieve reasonable curves, learning was artificially slowed through the addition of a rate parameter. So far, such a parameter has not been required in the current model, because there is much more to be learned. Chunks are still learned for light-button combinations, as in the earlier model, but these chunks are for only 1 of the 12 goals in the hierarchy in Figure 4-1 (not counting the recursive goals at 13 and 14). Additional chunks are required for the 11 other goals in the structure.

It is encouraging that the current model overcomes some of the minor flaws of the earlier system, but the major focus of this work is on the generality of the model. In addition to the Seibel task, the system so far has been applied to fourteen tasks from three different compatibility experiments [Fitts & Seeger, 1953; Morin & Forrin, 1962; Duncan, 1977]. As a sample of these results, Figure 7-2 shows a pair of simulated practice curves for two tasks from Fitts and Seeger [1953]. These curves contain fifty trials each, aggregated by five trials per data point.

This article is not the appropriate place to discuss the issues surrounding whether the simulated practice curves are truly power laws or something slightly different (such as an exponential). Suffice it to say that we obtain a mixture of exponential, power law, and ambiguous curves. These results roughly follow the predictions of the approximate mathematical analysis of the chunking theory appearing in Rosenbloom [1983]. They also fit with the observation that the human curves tend to be most exponential for the simplest tasks — the

¹³ At this point the *FranzLisp* system — which appeared to not be garbage collecting in the first place — refused to allocate any more memory for the simulation.

¹⁴ This data appear noisier than that in Figure 2-2 showing the human data from Seibel [1963]. This is accounted for by each point in Figure 2-2 being the mean of 1023 data points, rather than the 5 data points of this figure.

6.5 Chunking

As we saw in Section 5, chunking improves performance by enabling the system to use its experience with previous instances of a goal to avoid expanding the goal tree below it. In this section we detail how this has been implemented within the *Xaps3* architecture — yielding a working, task-independent, production-system practice mechanism. We begin by describing how chunks are used, and conclude with how they are acquired.

6.5.1 The use of chunks

The key to the behavior of a chunk, lies in its connection component. When a goal is proposed in a situation in which a chunk has already been created for it, the connection component of that chunk substitutes for the normal processing of the goal. This is accomplished in *Xaps3*, by hooking the connection to the desired state of the goal. The connection is a production containing a condition testing for the existence of a desired goal and a condition testing for the encoding of the goal's parameters¹⁰. It has two actions; one marks the goal as having succeeded, and the other asserts the goal's encoded result. If there is a desired goal in working memory in a situation for which a connection production exists, the connection will be eligible to fire. Whether (and when) it does fire, depends on conflict resolution. Connection productions are subject to the *parameter-passing bottleneck* conflict-resolution rule, because they pass the identifier of the desired goal as a parameter to the action that marks the goal as having succeeded. Therefore, only one goal can be connected at a time — part of the *bottleneck constraint* [Rosenbloom & Newell, In press] — and only one connection can execute for any particular goal instance (removing a source of possible error).

If the connection does fire, it removes the need to activate and expand the goal, because the goal's results will be generated directly by the connection (and decoding), and the goal will be marked as having succeeded. If instead, no connection production is available for the current situation of a desired goal, then eventually the production system will reach an impasse — no productions eligible to fire — and the goal will be activated and expanded. Therefore, we have just the behavior required of chunks — they replace goal activation and expansion, if they exist.

This behavior is, of course, predicated on the workings of the encoding and decoding components of the chunk¹¹. The encoding component of a chunk must execute before the associated connection can. In fact, it should execute even before the parent goal is activated, because the subgoal's encoded symbol must be part of the parent goal's initial state. Recall, that encodings are built up hierarchically. If the parent goal shares all of the parameters of one of its subgoals, then the encoding of the parent goal's parameters is partly based on the encoding generated for the subgoal. This behavior occurs in *Xaps3* because the encoding components are implemented as goal-free productions that do not pass parameters. They fire (concurrently with whatever else is happening in the system) whenever their parameters are in working memory (subject to conflict resolution). If the parameters exist before the parent

goal becomes active, as they must if they are parameters to it, then the encoded symbol becomes part of the parent goal's initial state.

As stated in Section 4, the decoding component must decode an encoded result when it will be needed. Each decoding component is a production that keys off of the non-primitive result pattern generated by the connection production, and off of an object of type *Decode*. When the architecture determines that decoding should occur, it places a *Decode* object in working memory, with the type of the object to be decoded specified by the *TYPE* attribute¹². The actions of the decoding production generate the component results out of which the non-primitive pattern was composed.

6.5.2 The acquisition of chunks

A complete specification of the chunk acquisition process must include the details of when chunks can be acquired, and from what information they are built. A chunk can be acquired when three conditions are met. The first condition is that some goal must have just completed. The system can't create a chunk for a goal that terminated at some point in the distant past, because the information on which the chunk must be based, is no longer available. Chunks also are not created prior to completion (on partial results). Chunks are simple to create in part because they summarize all of the effects of a goal. If chunks were created part way through the processing of a goal — for partial results of the goal — then a sophisticated analysis might be required in order to determine which parameters affect which results, and how. This is not really a limitation on what can be chunked, because any isolatable portion of the performance can be made into a goal.

The second condition is that the goal must have completed successfully. Part of the essence of goal failure, is that the system does not know why the goal failed. This means that the system does not know which parameter values have lead to the failure; thus, it can't create a chunk that correctly summarizes the situation. Chunking is success-driven learning, as opposed to failure-driven learning (see for example, Winston [1975]).

The third, and final, condition for chunk creation, is that all of the working-memory modifications occurring since the goal was first activated, must be attributable to that goal, rather than to one of its subgoals. This condition is implemented by assuring that no productions were fired while any of the goal's subgoals were active. All of the subgoals must either be processed by a chunk, or fail immediately after activation — failure of a subgoal, particularly of predicates, does not necessarily imply failure of the parent goal.

To summarize, a chunk is created after the system has decided to successfully terminate a goal, but before anything is done about it (such as marking the goal as succeeded, returning the results, and flushing the local objects from working memory). At that point the goal is still active, and all of its information is still available.

Most of the information on which the chunk is based, can be found in working memory (except, see below). The first piece of information needed for the creation of a chunk, is the name (and identifier) of the goal that is being chunked. This information is found by retrieving the object representing the active goal. The goal's explicit parameters are also available as attribute-value

¹⁰ Matters are somewhat more complicated for negated (<Not-Exists>) conditions [Rosenbloom, 1983].

¹¹ For efficiency, the encoding and decoding components are not created if there is only one parameter or result (respectively).

¹² Matters are actually somewhat more complicated [Rosenbloom, 1983].

next move (operator) may be problematic. Game playing programs effectively go into a choose-operator subgoal in which they simulate future paths in the game tree. They evaluate the alternatives, and return information allowing the selection of an operator in the top-level problem space.

Given such an augmentation to the current system, it would require only a small conceptual leap to imagine how chunking could be applied to these choose-operator subgoals. The resulting chunk would test those aspects of the situation that were examined during the evaluation of the operators, and yield the next operator (subgoal) to pursue. This is a search control chunk for the problem space — part of the backbone of the goal hierarchy.

One limitation of the current chunking mechanism that such a method-acquisition scheme could alleviate, is the inability of chunks to implement parameterized operators. Chunking always creates a totally specified piece of knowledge. As currently formulated, it cannot create the kinds of parameterized operators used as terminal nodes in the goal hierarchies (though it can achieve a form of separation of parameters¹⁶). On the other hand, problem spaces are inherently parameterized by their initial and desired states. Therefore, it may be that it is not necessary for chunks to create parameterized operators. These operators can come from another source (problem spaces). Chunks would only be responsible for making these operators more efficient.

In summary, doing method acquisition via chunking in problem spaces looks feasible, at least at the rather cursory level of detail at which it has been described here. If it does pan out, we hope to be able to formulate a number of the other difficult learning problems within this paradigm. The complications would appear as problem-solving in problem spaces, while the chunking mechanism would remain simple, merely recording the results generated by the problem-solving system.

10. Conclusion

At the beginning of this investigation we set out to develop a generalized, task-independent model of practice, capable of producing power-law practice curves. The model was to be based on the concept of chunking, and to be used as the basis (and a source of constraint) for a production-system practice mechanism. All of this has been accomplished. The generalized model that we have developed is based on a goal-structured representation of reaction-time tasks. Each task has its own goal hierarchy, representing an initial performance algorithm.

When a goal is successfully completed, a three-part chunk can be created for it. The chunk is based on the parameters and results of the goal. The encoding component of the chunk encodes the parameters of the goal, yielding a new symbol representing their combination. The connection component of the chunk ties the encoded parameter symbol to an encoded symbol for the results of the goal. The decoding component of the chunk, decodes the new result symbol to the results out of which it is composed.

¹⁶ One form of parameter independence achievable with the current system is the location-independent recognition of patterns. This can be accomplished by having one goal (or a set of goals) to recognize and relate the components of the pattern via their relative locations, and a second goal to determine the absolute location of the entire pattern. Each chunk for the first goal would be a location-independent pattern recognizer.

The chunk improves the performance of the system by removing the need to fully process the goal; the chunk takes care of it. The process of chunking proceeds bottom up in the goal hierarchy. Once chunks are created for all of a goal's subgoals in a specific situation, it is possible to create a chunk for the goal. This process proceeds up in the hierarchy until there is a chunk for the top-level goal for every situation that it could face.

Mechanisms for goal processing and chunking have been built into a new production-system architecture that fits within a set of constraints developed for the architecture of cognition. This architecture has been applied to a number of different reaction-time tasks (though not all of these results are presented here). It is capable of producing power-law practice curves.

As currently formulated, the chunking theory stakes out a position that is intermediary between three previously disparate mechanisms: classical chunking, memo functions, and production composition. These four ideas are different manifestations of a single underlying store-versus-compute idea.

And finally, we have outlined a research path by which the chunking theory, when integrated with a problem-solving system, can potentially be expanded to cover aspects of learning — such as method acquisition — outside of the domain of pure practice.

References

- Anderson, J. R. *Language, Memory, and Thought*. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1976.
- Anderson, J. R. Private communication. 1980.
- Anderson, J. R. Private communication. 1982.
- Anderson, J. R. Acquisition of cognitive skill. *Psychological Review*, 1982, 89, 369-406.
- Anderson, J. R. *The Architecture of Cognition*. Harvard Press, 1983.
- Bower, G. H. Perceptual groups as coding units in immediate memory. *Psychonomic Science*, 1972, 27, 217-219.
- Bower, G. H., & Springston, F. Pauses as recoding points in letter series. *Journal of Experimental Psychology*, 1970, 83, 421-430.
- Bower, G. H. & Winzenz, D. Group structure, coding, and memory for digit series. *Journal of Experimental Psychology Monograph*, 1969, 80, 1-17. (May, Pt. 2).
- Card, S. K., English, W. K. & Burr, B. Evaluation of mouse, rate controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics*, 1978, 21, 601-613.
- Chase, W. G., & Ericsson, K. A. Skilled memory. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1981.
- Chase, W. G. & Simon, H. A. Perception in chess. *Cognitive Psychology*, 1973, 4, 55-81.
- Crossman, E. R. F. W. A theory of the acquisition of speed-skill. *Ergonomics*, 1959, 2, 153-166.
- DeGroot, A. D. *Thought and Choice in Chess*. The Hague: Mouton, 1965.
- Duncan, J. Response selection rules in spatial choice reaction tasks. In S. Dornic (Ed.), *Attention and Performance VI*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.
- Ernst, G. W. & Newell, A. *GPS: A Case Study in Generality and Problem Solving*. New York: Academic Press, 1969. (ACM Monograph).

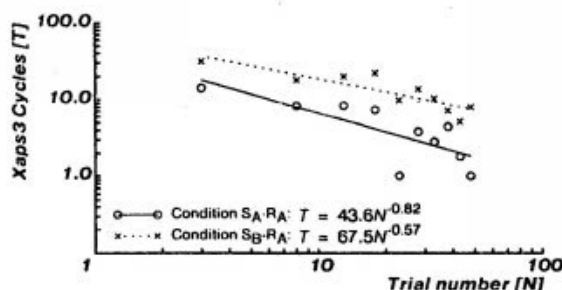


Figure 7-2: Simulated practice curves for conditions S_A-R_A and S_B-R_A from Fitts and Seeger [1953]. The latter curve is the average over two hierarchy variations.

compatibility tasks are among the simplest tasks for which we have human practice data. For more on this issue, see Newell and Rosenbloom [1981] and Rosenbloom [1983].

8. Relationship to Previous Work

In addition to the already mentioned relationship to caching, the current formulation of the chunking theory of learning provides an interesting point of contact between three previously disparate concepts: (1) classical chunking, (2) memo functions [Michie, 1968], and (3) production composition [Lewis, 1978; Neves & Anderson, 1981; Anderson, 1982b]. Classical chunking has already been described in Section 3. A memo function¹⁵ is a function with an auxiliary table added. Whenever the function is evaluated, the table is first checked to see if there is a result stored with the current set of parameter values. If so, it is retrieved as the value of the function. Otherwise, the function is computed and the arguments and result are stored in the table.

Production composition is a learning scheme whereby new productions are created through the combination of old ones. Given a pair of productions that execute successively, their composition is created from their conditions and actions. The condition side of the new production consists of all of the conditions of the first production, plus those conditions from the second production that do not match actions of the first production. The action side consists of all of the actions from both of the productions. The resulting composition is a single production that accomplishes the combined effects of the older pair of productions.

Though these mechanisms were proposed in quite different contexts, and bear little obvious relationship to each other, the current formulation of the chunking theory of learning has strong ties to all three. It (1) explains how classical chunks can be created and used; (2) caches the results of computations, as do memo functions; and (3) results in productions similar to those generated by goal-directed production composition [Sauers & Farrell, 1982; Anderson, 1983]. The chunking theory differs from these three mechanisms in a number of ways [Rosenbloom, 1983], but at the same time it occupies a common ground

¹⁵ Memo functions themselves are derived from the earlier work by Samuel [1959] on creating a rote memory for the values of board positions in the game of checkers.

between them. This leads us to propose that all four mechanisms are different manifestations of a single underlying store-versus-compute idea. We feel that the chunking theory has a lot going for it, but it is probably too early to know what formulation will work out to be the correct one in the long run for a general practice mechanism.

9. Expanding the Scope of Chunking

In this work we have shown how chunking can provide a model of practice for tasks that can already be accomplished. Performance is sped up, but not qualitatively changed. It is interesting to ask whether chunking can be used to implement any of the other more complex forms of learning — such as method acquisition, concept formation, generalization, learning by being told, and expectation-driven learning. Chunking does not directly accomplish any of these forms of learning, and on first glance, the table look-up nature of chunking would seem to preclude its use in such sophisticated ways. However, we would like to at least raise the possibility that the scope of chunking may extend much further.

Method acquisition is the prototypical case of "hard" learning. Given an arbitrary task, how does the system first construct a method (goal structure) for it? The answer may lie in the integration of chunking with problem-space search. The *problem space hypothesis* [Newell, 1980b; Laird & Newell, 1983] states that intelligent agents are always performing in a problem space. At any instant, the agent will have a goal that it is attempting to fulfill. Associated with that goal is a *problem space* in which the goal can be pursued. The problem space consists of a set of *states*, a *problem* (initial and desired states), a set of *operators* that move the agent between states, and *search control* information which assists in guiding the agent efficiently from the initial to the desired state.

One way to look at the goal structures used in the current model is as degenerate problem spaces. The goals are certainly there as needed. The current state can be identified with the contents of working memory, with the set of possible states being defined as all possible working memories. The initial state for a goal is its initial state in the production system. The desired state is defined by the production that tests for goal success. The operators are the subgoals — an operator is simply a goal that the system knows how to accomplish. The search control consists of those productions that specify which subgoals to do when. The one thing missing is the ability to search — the essence of working in a problem space.

Adding a search capability to the current system could be the first step towards method acquisition. It would allow the system to perform tasks that it does not already "know" how to do. The second step must be to show how chunking could lead from an inefficient search, to the development of an efficient goal-structured method.

This may go as follows. Suppose the system is at some state for which it does not know how to make further progress towards the desired state. What should it do? It should create a subgoal in which it can figure out which operator to do next. This subgoal could be pursued in any number of ways. One of the simplest is to ask someone else what to do next (leading to learning by being told). Another possibility is that a look-ahead search can be performed. If a path is found to the desired goal, the first operator on that path is returned as the result of the search. This is of course the paradigm case for a game-tree search, in which the goal is to win the game, but choosing the

Learning Physical Domains: Towards a Theoretical Framework

Kenneth D. Forbus, Dedre Gentner
Bolt, Beranek and Newman, Inc.
10 Moulton Street
Cambridge, Mass. 02139, USA

Abstract

We describe a theoretical framework being developed in an attempt to account for human learning of physical domains. We are using Qualitative Process Theory to model people's physical knowledge and the Structure-Mapping Theory to characterize the computations that move a learner from one representation to another. Here we outline the component theories and propose a learning sequence for physical domains.

1. Introduction

People use and extend their knowledge of the physical world constantly. Understanding how this fluency is achieved would be an important milestone in understanding human learning and intelligence. Our primary purpose is to construct a computational account of human learning.

Below we describe the theoretical framework we are developing in an attempt to account for human learning of physical domains. Any such framework must describe the form of the domain models and the kinds of computations that move the learner from one representation to another. Thus we first briefly summarize Qualitative Process Theory [Forbus, 1981, 1982] and the Structure Mapping Theory [Gentner, 1980, 1983], the component theories that underlie our framework. We then describe a learning sequence, composed of four stages of physical models, and our conjectures about how a learner constructs one type of theory from another.

2. Qualitative Process Theory

Our first requirement is a language in which to describe people's commonsense knowledge about physical situations. People know about a great many kinds of physical changes: things move, collide, bend, break, heat up, cool down, flow and boil. Intuitively we think of these as *processes*. Qualitative Process theory attempts to formalize this notion. An important property for our purposes is that both correct and incorrect physical models can be expressed.

We view a physical situation as composed of objects and relationships among them. The continuous parameters of an object, such as temperature and pressure, are represented by *quantities*. In QP theory the value of a number is defined in terms of its *Quantity Space* - a collection of inequalities which hold between it and other quantities. The Quantity Space is a useful qualitative representation because processes typically start and stop when inequalities between parameters change.

Processes cause changes by *influencing* quantities. There are two kinds of influence which can occur. A quantity can be *directly influenced* by a process or processes, or indirectly influenced because it is a function of some other quantity that is influenced. We will specify functional dependencies between quantities by

$$Q1 \propto_Q Q2$$

(read "Q1 is qualitatively proportional to Q2"), meaning there exists a

function which determines Q1 and is increasing monotonic in its dependence on Q2. $\propto_Q$ expresses partial information, since the implicitly specified function may or may not depend upon other quantities. Figuring out the sign of a quantity's derivative from its influences is called *resolving* its influences.

A process is a thing that acts through time to cause changes. Processes are specified by five parts:

Individuals: Descriptions of the objects which participate in the process.

Quantity Conditions: Inequality statements and status assignments to other conditionalized descriptions which must be true for the process to be active.

Preconditions: Statements other than Quantity Conditions that must be true for the process to be active.

Relations: The relationships between the individuals which hold when the process is active.

Influences: Descriptions of what quantities are directly influenced by the process.

A process acts between any collection of objects which match the individual specifications, whenever both the Preconditions and Quantity Conditions are true. Preconditions, as distinct from Quantity Conditions, express changes that cannot be predicted solely within QP theory - physics alone will not tell us that someone will walk by and turn on a stove, for instance, although it can predict that a result of this action will be that the water sitting in the kettle on top of it will soon boil. The Relations specify the functional dependencies induced by the process as well as any new entities which it introduces by virtue of being active, such as a gas generated by boiling.

The collection of processes acting at any time characterizes "what is happening" then in that situation. A reasoner's theory of dynamics for a domain is characterized in terms of a *Process Vocabulary* that describes the kinds of processes the reasoner believes can occur. The actual changes that occur as a result of processes acting are represented as a *history* [Hayes, 1979].

QP theory sanctions several basic deductions. These include determining what processes might occur, what processes are occurring, and determining how the set of active processes can change over time. These basic deductions can be combined to perform more complex reasoning tasks. Two examples of more complex deductions are qualitative simulation [Forbus, 1982] and measurement interpretation [Forbus, 1983].

3. Comparisons and Structure Mapping

So far we have considered how a Naïve Physics might be represented. We now turn to the question of how such domain models might be learned. A major process in experiential learning is comparing the current situation with stored descriptions. Consider a person who has just moved to a cold climate learning to operate his furnace. By the end of the winter he may learn that he cannot cause the house to get warm any sooner by setting the thermostat higher than the desired temperature. There are at least three different ways

- Fitts, P. M., & Seeger, C. M. S-R compatibility: Spatial characteristics of stimulus and response codes. *Journal of Experimental Psychology*, 1953, 46, 199-210.
- Forgy, C. L. *OPS4 User's manual* (Tech. Rep. CMU-CS-79-132). Carnegie-Mellon University Department of Computer Science, July 1979.
- Johnson, N. F. Organization and the concept of a memory code. In Melton, A. W. & Martin, E (Eds.), *Coding Processes in Human Memory*. Washington, D.C.: Winston, 1972.
- Kolers, P. A. Memorial consequences of automatized encoding. *Journal of Experimental Psychology: Human learning and memory*, 1975, 1(6), 689-701.
- Laird, J. E., and Newell, A. A universal weak method: Summary of results. In *Proceedings of the Eighth IJCAI*, 1983.
- Lewis, C. H. *Production system models of practice effects*. Doctoral dissertation, University of Michigan, 1978.
- McDermott, J. R1: a rule-based configurator of computer systems. (Tech. Rep. 80-119). Carnegie-Mellon University Computer Science Department, 1980.
- Michie, D. "Memo" functions and machine learning. *Nature*, 1968, 218, 19-22.
- Miller, G. A. The magic number seven plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 1956, 63, 81-97.
- Moran, T. P. *Compiling cognitive skill* (AIP Memo 150). Xerox PARC, 1980.
- Morin, R. E., & Forrin, B. Mixing two types of S-R associations in a choice reaction time task. *Journal of Experimental Psychology*, 1962, 64, 137-141.
- Neisser, U., Novick, R., Lazar, R. Searching for ten targets simultaneously. *Perceptual and Motor Skills*, 1963, 17, 427-432.
- Neves, D. M. & Anderson, J. R. Knowledge compilation: Mechanisms for the automatization of cognitive skills. In Anderson, J. R. (Ed.), *Cognitive Skills and their Acquisition*. Hillsdale, NJ: Erlbaum, 1981.
- Newell, A. Heuristic programming: Ill-structured problems. In Aronofsky, J. (Ed.), *Progress in Operations Research, III*. New York: Wiley, 1969.
- Newell, A. Production systems: Models of control structures. In Chase, W. G. (Ed.), *Visual Information Processing*. New York: Academic Press, 1973.
- Newell, A. Harpy, production systems and human cognition. In Cole, R. (Ed.), *Perception and Production of Fluent Speech*. Hillsdale, NJ: Erlbaum, 1980. (Also available as CMU CSD Technical Report, Sep 1978).
- Newell, A. Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson (Ed.), *Attention and Performance VIII*. Hillsdale, N.J.: Erlbaum, 1980. (Also available as CMU CSD Technical Report, Aug 79).
- Newell, A. & Rosenbloom, P. S. Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, NJ: Erlbaum, 1981.
- Newell, A. & Simon, H. A. *Human Problem Solving*. Englewood Cliffs: Prentice-Hall, 1972.
- Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence*. New York: McGraw-Hill, 1971.
- Rosenbloom, P. S. *The Chunking of Goal Hierarchies: A Model of Practice and Stimulus-Response Compatibility*. Doctoral dissertation, Carnegie-Mellon University, 1983.
- Rosenbloom, P. S., & Newell, A. Learning by chunking: Summary of a task and a model. In *Proceedings of AAAI-82, National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, 1982.
- Rosenbloom, P. S. & Newell, A. Learning by chunking: A production-system model of practice. In D. Klahr, P. Langley, & R. Neches (Eds.), *Self-Modifying Production System Models of Learning and Development*. MIT Press, In press. (Also available as Carnegie-Mellon University Computer Science Tech. Rep. # 82-135).
- Samuel, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 1959, 3, 210-229.
- Sauers, R., and Farrell, R. *Grapes user's manual*. 1982.
- Seibel, R. Discrimination reaction time for a 1,023-alternative task. *Journal of Experimental Psychology*, 1963, 66(3), 215-226.
- Shortliffe, E. H. *Computer-Based Medical Consultations: MYCIN*. New York: American Elsevier, 1976.
- Snoddy, G. S. Learning and stability. *Journal of Applied Psychology*, 1926, 10, 1-36.
- Winston, P. H. Learning structural descriptions from examples. In Winston, P. H. (Ed.), *The Psychology of Computer Vision*. New York: McGraw Hill, 1975.

will(has) happen(ed)."

Consider a balance beam or see-saw. If we place a weight on each side of the fulcrum, it will either tilt counter-clockwise, clockwise, or not at all. Most people have had enough experiences with see-saws to have formed protohistories concerning their behavior. By our conjecture, a protohistory is automatically available whenever we encounter a see-saw. From it, we can often predict which way the particular see-saw will move. For example, we may have a protohistory that describes what happens if a small person gets on opposite a large person.

However, the predictive power of protohistories is quite limited. There is no guarantee that the features matched actually correspond to relevant factors. For example, we will be fooled when a large person sits close to the fulcrum if we have formed our see-saw protohistory from watching people sitting at equal distances. Massive overlap in features is needed for reliable use, which means protohistories will yield conclusions in fewer situations than a true theory would. Furthermore, there is no certain way to decide between conflicting results if more than one protohistory matches a situation.¹

4.1.1 Learning Protohistories

The process of constructing protohistories requires dividing up experience into classes according to literal similarity and abstracting a summary for each class. There has been little direct research on this process. However, investigations into the process of constructing object prototypes provides some hints. First, people seem to be able implicitly to compute a kind of component match. Second, this intersection is not merely a simple feature intersection; rather, it appears that configurations among features are important in the prototype. Third, once this prototype is computed, it has powerful effects on subsequent processing of experience. Once people abstract a prototype from a set of patterns, they may be more confident of having seen the prototype (which was never presented) than they are of having seen the actually presented patterns (Posner & Mitchell, 1967). Finally, people may not be aware of forming prototypes, except as a general sense of increased familiarity with a category.

To summarize, if protohistories behave like object prototypes, then we should find that they are computed implicitly; that they are sensitive to configurations among events; that, once computed, they show the recognition strength and other psychological privileges of prototypes.

4.2 Stage 2 - The Causal Corpus

While protohistories summarize the phenomena, they are not a theory of it. Jumping directly to a detailed theory can be quite difficult. For example, the space of possible equations connecting all observable parameters in a typical situation can be quite large. Instead, we conjecture that weaker theories are formed first, theories that characterize which parts of the situation are relevant to desired

conclusions. In particular, we conjecture that a collection of CAUSE statements, the *Causal Corpus*, is computed from prototype objects and protohistories.

We view CAUSE as an approximation concept, in particular a weak form of ontological commitment. Saying

CAUSE(A, B)

means that there is a mechanism, specified by some theory T, such that

$A \wedge T \Rightarrow B$

Processes provide the notion of mechanism for physical domains. The statement

CAUSE(Triangle(F), Has-Three-Sides(F))

is not a legitimate use of CAUSE by this definition, since the required axioms of geometry do not specify a mechanism.

The further distillation of experience from protohistories to the Causal Corpus serves two purposes (aside from its potential as a means of data reduction). First, it provides a collection of heuristics that can be used directly to draw inferences. While the learner doesn't have firm grounds to consider the CAUSE statements complete or correct, often they may suffice for the desired class of inferences. Second, the collection of CAUSE relations can be used to guide the search for a deeper theory of the domain. The CAUSE statements suggest connections between various aspects of the domain which a deeper theory must either explain or explain away.

Returning to our see-saw example, we might attempt to apply our Causal Corpus to a balance beam built out of blocks. Suppose we call the two blocks on it A and B. Our Causal Corpus might be:

Cause(Bigger(A, B), Tilt-Towards(A))
Cause(Farther(A, B), Tilt-Towards(A))

These statements can be interpreted as rules in several ways: if we see that block A is bigger than block B we can predict tilt, and if we see tilt we may hypothesize that one block is farther out than another. These statements are more broadly applicable than protohistories since they refer to fewer properties. Unlike protohistories, the Causal Corpus can be debugged to some degree.

However, the approximate nature of the CAUSE relation limits the ability to discriminate between conflicting predictions. With the Causal Corpus above, for instance, if block A is bigger and block B is farther out we will have two predictions. As we shall discuss, in order to make more precise inferences the learner must eventually uncover the mechanisms whose behavior the Causal Corpus describes.

4.2.1 Learning the Causal Corpus

We suspect there are three techniques for computing and debugging a Causal Corpus. The first technique is to hypothesize causality from co-occurrence:

If you always see A before B,
then hypothesize CAUSE(A, B)

and

If A is true whenever B is true,

1. There are of course heuristic criteria, such as taking the one that matches "best" or the one which has worked most often. The problem with such heuristics is that little is learned from mistakes.

he could reach this conclusion. First, he may compare the current situation with past furnace experiences and notice a regularity. Second, he may match the current situation with a known abstraction, such as a position-action controller (as opposed to a proportional-action controller), to deduce the answer. Third, he may use an analogy, comparing the current situation with a description from another domain, such as fluid flow, to suggest governing principles. Each of these ways of learning relies on some form of comparison, either with a stored record of literally similar events, with a stored abstraction, or with a stored description that can function as an analogy.

The *Structure Mapping Theory* concerns such comparisons (Gentner, 1980, 1981, 1983; Gentner & Gentner, 1983). The theory describes the rules that are used to import a descriptive structure from one domain (the *base domain*) to another (the *target domain*). Literal Similarity, Analogy, and Abstraction Mappings (applications of general laws) are viewed as different kinds of mappings between descriptions. The types of comparisons are defined syntactically, and depend only on the form of the knowledge representation, not its content. In particular:

- (1) A *literal similarity* statement is a comparison in which a large number of predicates is mapped from base to target. There are many more shared predicates than nonshared predicates. The mapped predicates include both object-attributes and relational predicates.¹
- (2) An *analogy* is a comparison in which relational predicates, but few or no object attributes, can be mapped from base to target. The particular relations mapped are determined by *systematicity*, as defined by the existence of higher-order constraining relations.²
- (3) An *abstraction mapping* is a comparison in which the base domain is an abstract relational structure. Predicates from the abstract base domain are mapped into the target domain. Since there are no object attributes in the base, there are no one-place predicates to be left behind. Applying a rule to a situation is an example of the abstraction mapping. Sometimes we will also refer to the relational structure so mapped as an abstraction.

The *Analogical Shift Hypothesis* concerns how these comparisons are used in learning and reasoning about a domain. In the earliest stages most of the comparisons are literal similarity matches. As the domain becomes familiar, analogies begin to occur. Successful analogies may yield abstractions. Analogy serves as a means of changing mental models, and therefore plays an important role in the middle stages of learning. In the later stages of learning, abstraction mappings and literal similarities predominate.

The role of a comparison in learning depends on at least two things: (1) *accessibility*, the likelihood that the match will be noticed; and (2) *usefulness*, what can be deduced from the match, if it is

accessed. Accessibility, in turn, depends at least on (i) the *familiarity* of the base description and (ii) the overall *similarity* between the base description and the current target. The comparisons discussed above behave very differently with respect to these two criteria.

Literal similarity matches are quite accessible, but of limited usefulness. Psychological evidence indicates that people automatically perform such comparisons to combine perceptually similar experiences into composite prototypes. However, such composites are of limited use in revealing the principles that govern the phenomenon. In general, literal similarity matches are highly accessible, since they can be indexed by object descriptions, by relational structures, or by both. But they are not very useful in deriving causal principles, precisely because there is too much overlap to know what is crucial. Abstractions are extremely useful, but not necessarily accessible: novice learners may not know the appropriate abstraction, or it may be so unfamiliar that its accessibility will be low. Thus abstraction mappings, while ultimately important, are unlikely to play a major role in the early stages of learning.

Analogies lie between the highly accessible literal similarity matches and the highly useful abstraction matches. Potential analogies are less accessible than literal similarity matches, since they require accessing the learner's data base via relational matches; object matches are of no use. However, once found, an analogy should be more useful than a literal similarity match in deriving the key principles, since the shared data structure is sparse enough to permit analysis. Moreover, if we assume the systematicity principle, then the set of overlapping predicates is likely to include higher-order relations, such as causality. Thus analogy can function to reveal principles in a domain that previously lacked the appropriate abstractions (Gick & Holyoak, 1981). Winston's (1981) program, which derives if-then rules by abstracting the predicates common to two analogs, is a case in point.

4. Stages of Understanding

We suspect there are four kinds of mental models that will be generated in the process of understanding. The sequence of models we propose is developmental in that the theories of each stage are generated both by the phenomena being understood and the theories of the stage before it. We do not think every person goes through each stage for every domain, nor that a person is at the same stage in every domain at the same time.

4.1 Stage 1 - ProtoHistories

Suppose we are observing some new physical phenomena. All we can do is observe and remember what is happening. We conjecture that the simplest physical models of a domain are *protohistories* -- prototype histories which serve as summaries of experience.¹ Like object prototypes, protohistories are the "most typical instances" of phenomena (see Rosch, 1973, 1975, 1976; Smith & Medin, 1981). The terms in these descriptions are observables, and their deductive import can be roughly expressed as "If I see X, then Y

1. Object attributes are predicates which take one object as an argument, such as RED(x). Relations are predicates which take two or more arguments, such as COLLIDE(x, y).

2. We define the *order* of a proposition as follows: Constants have order zero, as do functions on them. The order of a proposition is one plus the maximum of the orders of its arguments. Thus COLLIDE(x, y) would be first order if x and y are domain objects, and CAUSE(COLLIDE(x, y), BREAK(x)) would be order two. Examples of higher-order relations are CAUSE and IMPLIES.

1. Some of diSessa's *Phenomenological Primitives* appear to be representable as protohistories [diSessa, 1983].

which specify useful default resolutions for the ambiguities inherent in qualitative descriptions. Examples of these ambiguities include knowing circumstances in which certain processes can be ignored (such as evaporation when pouring water from one glass to another) and what their net effect is (such as Black's law for the temperature of mixtures).

Returning to the balance beam example, we know that the force of a block on the beam is qualitatively proportional to the mass of the block and to the distance from the fulcrum. If we also know that the force is the product of distance and weight, then providing numerical values for these quantities will allow an unambiguous prediction about tilt.

4.4.1 Learning Expert Models

One part of refining our commonsense models is to elaborate α_q statements into constraint equations. [Langley, 1979] describes techniques that should be useful.

Developing rules of thumb requires discovering which outcomes raised by qualitative reasoning are unlikely and which potential interactions can be ignored. We suspect the possible behaviors raised by Naive Physics are compared against known protohistories. Hypothesized outcomes that have no corresponding protohistory are considered unlikely, while those corresponding to a very familiar protohistory are considered very likely.

5. Summary

We have described our progress in weaving together the Structure Mapping theory and Qualitative Process theory into a framework that aims to account for learning in physical domains. We are still at the stage of refining the questions, rather than providing answers.

6. Acknowledgements

This project is carried out at Bolt, Beranek, and Newman, Inc., under contract to Schlumberger-Doll. We thank Allan Collins, Albert Stevens, and Bob Berwick for their insightful comments.

7. References

- de Kleer, J. Causal and Teleological Reasoning in Circuit Recognition TR-529, MIT AI Lab, Cambridge, Massachusetts, September 1979
- diSessa, A. Phenomenology and the Evolution of Intuition. in Mental Models, D. Gentner and A. Stevens, LEA, Inc, April 1983
- Forbus, K. Qualitative Reasoning about Physical Processes. Proceedings of IJCAI-7, 1981
- Forbus, K. Qualitative Process Theory. MIT AI Lab Memo No. 664, 1982
- Gentner, D. The structure of analogical models in science (BBN Report No. 4451). Cambridge, Mass.:Bolt Beranek and Newman Inc., 1980.
- Gentner, D. Structure-mapping: A theoretical framework for analogy. Cognitive Science, 1983, 7, No. 2.
- Gick, M. L., & Holyoak, K. J. Analogical problem solving. Cognitive Psychology, 1980, 12, 306-355.
- Hayes, P. J. The Naive Physics Manifesto in Expert Systems in the Micro-Electronic Age, edited by D. Michie, Edinburgh University press, May 1979
- Hayes, P. J. Naive Physics 1 - Ontology for Liquids Memo, Centre pour les etudes Semantiques et Cognitives, Geneva, 1979
- Langley, P. Rediscovering Physics with BACON.3 Proceedings of IJCAI-6, 1979
- Posner, M. I., & Gentner, D. R. Chronometric analysis of classification. Psychological Review, 1967, 74, 392-409.
- Sussman, G. A Computational Model of Skill Acquisition. Springer-Verlag, New York, 1976
- Winston, P. Learning new principles from precedents and exercises. MIT AI Lab Memo No. 632, May 1981

then hypothesize CAUSE(A, B)

These rules make certain assumptions on the form of memory, namely that some number of circumstances can be remembered, and in sufficient detail so that A is either explicitly stored or computable from what is stored. Protohistories should serve as a means of data reduction, reducing the complexity of constructing a Causal Corpus with these rules.

We also conjecture a class of rules which are used to debug a Causal Corpus in the face of new information (c.f. [Sussman, 1976]). Each rule corresponds to a hypothesis about a bug in the structure of the Causal Corpus, such as a missing precondition. We have not addressed the issue of judging a Causal Corpus for consistency and completeness; such criteria will play a major role in controlling the debugging rules and in the mixture of generation and debugging that occurs.

Analogy provides the third technique for extending a Causal Corpus. The CAUSE relations from one domain can be mapped into another, since CAUSE qualifies as a higher-order constraining relation (see [Winston, 1981]).

4.3 Stage 3 - Naive Physics

The Naive Physics models replace CAUSE statements with theories about the specific mechanisms of change. The ontology is extended by adding processes to explain observed changes. The ontology also includes properties and objects that are not directly observable (for example, heat and heat flow) and the new relationships (such as fluid path and heat path) required to reason about them.

An important advantage of these models is the ability to reason by exclusion. Unlike the previous stages, predictions that fail still yield information about the situation. For instance, if fluid is seen flowing into a container and the level is not rising, then it is reasonable to assume that fluid is flowing out of it through some unknown path.

Returning to our balance beam, we might use a process SWING to describe rotation around a contact point. The Precondition would be that the object is free to rotate, and the Quantity Condition would be a non-zero angular velocity. SWING directly influences the angular position of the beam. Further, its influence is qualitatively proportional to the angular velocity. Thus a prediction concerning tilt becomes a prediction about which instance, if any, of the SWING process will occur. Net force becomes the crucial parameter; to deduce which way something swings requires comparing the forces exerted on each side of the beam. The factors illuminated in the Causal Corpus of "Bigger" and "Farther" become the quantities "Mass" and "Distance". Examining the description of SWING and the CAUSE statements suggests we write force as a function of the two parameters, and in particular:

$$\begin{aligned}\text{Force}(A) &\propto_0 \text{Distance}(A, \text{pivot}) \\ \text{Force}(A) &\propto_0 \text{Mass}(A, \text{pivot})\end{aligned}$$

which brings us one step closer to a model that can always determine which way something will tilt.

4.3.1 Learning Naive Physics

The major problem in learning a naive physics is constructing a vocabulary of processes which consistently describes experience. The learner must strip away the irrelevant predicates that are part of his protohistories and Causal Corpus; and often, he must construct new objects and properties that are not observable.

The Causal Corpus provides a search space for potential Process Vocabularies. Each statement in the CC must be elaborated into a consequence of a Process Vocabulary. There appear to be only a small number of distinct ways to perform the elaboration depending on the particular form of the arguments. For example, the statement

The decrease in amount of Q
causes the level to fall.

indicates that some active process (or Individual View) in the situation contains the statement

$$\text{Level}(Q) \propto_0 \text{Amount-of}(Q)$$

in its Relations.

While hypothesizing a Process Vocabulary on the basis of the Causal Corpus should be much simpler than working directly from protohistories or direct observation, it still appears difficult. We conjecture several constraints that make the problem more tractable. First, we think people are conservative in the introduction of unobserved properties. For example, we have interviewed subjects whose model of a domain appears to be organized around one parameter - a "generalized strength" attribute - which is used in all Quantity Conditions and comparisons. In reasoning about fluids, for instance, pressure, flow rate, and velocity appear to be used as different names for the same thing. In electricity, voltage, current, power, potential, and velocity of electrons are used interchangeably by some people. The advantage of this generation strategy is of course that simpler models will be explored first, with further distinctions made only when necessary. Second, some physical laws can be viewed as constraints on what Process Vocabularies are possible. Conservation of Energy, for example, demands that if a process directly influences a quantity that represents some kind of energy it must also directly influence some other quantity which represents energy in the opposite direction.

Once again, Analogy can provide a constructive mechanism. It can be used to import candidate processes from previously understood domains, such as understanding heat flow in terms of fluid flow.

4.4 Stage 4 - Expert Models

The models generated so far still contain fundamental ambiguities.¹ The final stage of learning consists of discovering ways to resolve those ambiguities. We suspect several kinds of knowledge are involved. (Obviously, the potential complexity of the models is open-ended.) These kinds of knowledge include equations to describe relationships between parameters and "rules of thumb"

1. [de Kleer, 1979] discusses the nature of ambiguity in qualitative descriptions.

P1&P2

```

IF the goal is to: dial = telephone-number
  and =digit1 is the first digit of =telephone-number
  and =digit2 is after =digit1 in =telephone-number
THEN dial =digit1 and then =digit2.

```

Compositions like this will reduce the number of production applications to perform the task.

A composed production like P1&P2 still requires that the information (in this case, the telephone number) be retrieved from long-term memory, held in working memory, and matched to the second and third clauses in P1&P2. Proceduralization eliminates clauses in the condition of a production that require information to be retrieved from long-term memory and held in working memory. In P1&P2, the second and third condition clauses would be eliminated. The variables that would have been bound in matching these clauses are replaced by the values that they are bound to in the special case. If this production is applied in dialing Mary's telephone number, which is 432-2815, the variables in P1&P2 would be bound as follows:

```

=telephone-number = Mary's number
  =digit1 = 4
  =digit2 = 3

```

Substituting these values for the variables and eliminating the second and third condition clauses transforms the production into:

P1&P2*

```

IF the goal is to dial Mary's number
THEN dial 4 and then 3.

```

By further composition and proceduralization, a production can be built that dials the full number:

P*

```

IF the goal is to dial Mary's number
THEN dial 4-3-2-2-8-1-5.

```

It should be emphasized that forming this production does not necessarily imply the loss of the declarative representation nor of the ability to use it interpretively. In the few reported cases of loss of declarative access to a telephone number, the declarative knowledge has ceased to be used and has simply been forgotten.

Elsewhere (Anderson, 1982, 1983a; Neves & Anderson, 1981) we have discussed the evidence for knowledge compilation from the literature of experimental psychology. The major issue to be considered here is how these mechanisms are implemented and used in relatively complex problem-solving domains. Our work on this topic has been done in the context of the GRAPES simulation of the ACT theory (Sauers and Farrell, 1982). GRAPES is a system devoted to simulating ACT in the context of novice LISP programming. It simulates the problem solving and programming of novices in writing LISP functions and how novices learn from their problem-solving episodes. I will describe here first a typical GRAPES simulation of a subject and then focus on the mechanisms of knowledge compilation underlying that simulation.

A Simulation of LISP Programming

One of our consistent observations about novices is that they are not able to read instructions, of even modest complexity, and then generate, without error, the behavior instructed. This is not surprising given the ACT theory. According to that theory, instructions are stored initially in a declarative form while behavior requires procedures which are represented as productions. Instructions cannot directly set up procedures to perform the skill. To get behavior, general interpretive productions must convert this knowledge into behavior. Many of the problems arise because of the indirection through these interpretive productions.

I will discuss one of the early learning episodes that we have simulated. If the reader is interested in the simulation of more complex programming behavior he is referred to Anderson, Farrell, and Sauers (1982). The problem-solving episode to be discussed, from one of our subjects BR, is typical of the difficulties people have in making the transition from instruction to experience. She had read the instruction on pages 33-37 of Winston and Horn (1981) on function definition and had turned to the first problem on page 37 to define the function FIRST. This function was to return the first element of a list. This was her first attempt at defining a function.

She extracted virtually nothing from the text instruction. What she did extract was a template for how to write a function definition:

```

(DEFUN <function name>
  (<parameter 1> <parameter 2>...<parameter n>)
  <process description>)

```

Winston and Horn assert "angle brackets delineate descriptions of things". She also studied three examples of function definitions to which she referred. The first converted Fahrenheit to centigrade:

```

(DEFUN F-TO-C (TEMP)
  (QUOTIENT (DIFFERENCE TEMP 32) 1.8))

```

The second exchanged the first two members of a list:

```

(DEFUN EXCHANGE (PAIR)
  (LIST (CADR PAIR) (CAR PAIR)))

```

The third returned the percentage by which the second argument is larger than the first:

```

(DEFUN INCREASE (X Y)
  (QUOTIENT (TIMES 100.0
    (DIFFERENCE Y X))
    X))

```

The one other relevant thing BR knew was the function CAR and how to use it when interacting with the monitor in LISP. CAR returns the first element of the list which is its argument. She knew, for instance, that if she types (CAR '(A B C)), the monitor would return the answer A. Thus, this problem is really an exercise in using the syntax of function definition rather than defining a novel function.

BR's method of solving this problem relied heavily on trying to use the structure of the template and the examples to guide her function writing. We refer to this process as *structural analogy*. We created a production system in GRAPES that would simulate this protocol. The only productions we required for this simulation were

Knowledge Compilation: The General Learning Mechanism

John R. Anderson
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

The ACT learning mechanisms of knowledge compilation are discussed. Knowledge compilation consists of the mechanisms of composition and proceduralization. Composition operates by collapsing multiple productions into a single production that has the effect of the set. Proceduralization operates by building into productions information that previously had to be retrieved from long-term memory. It is shown how these two mechanisms can simulate the initial stages of skill acquisition in the domain of learning how to program in LISP. It is also shown that these mechanisms can reproduce the effects of the generalization and discrimination learning mechanisms. Generalizations and discriminations emerge as the consequence of compiling the processes of analogy formation and error correction.

Key terms: learning, knowledge compilation, composition, proceduralization, skill acquisition, simulation of LISP programming, analogy, generalization, discrimination, cognitive modelling.

Introduction

For a number of years (Anderson, 1982, 1983a; Anderson, Sauer, & Farrell, 1982; Neves & Anderson, 1981) we have been working on a general set of learning principles that come under the rubric of *knowledge compilation*. These principles are concerned with how a new skill is acquired such as generating a proof in geometry (Anderson, 1983b). The general framework is one in which a learner is viewed as beginning with declarative information relevant to the execution of a skill. For instance, in the case of geometry, the student might learn about the properties of two column proofs and various theorems and postulates. This information is stored in declarative form--i.e., as facts about the domain. For this knowledge to be used, general *interpretive* procedures must apply to use these facts. Two common types of interpretive procedures are general problem-solving procedures and general analogy procedures. Knowledge compilation operates on the traces of such procedures. It will create new procedures which are specific to that domain and which are more efficient.

This paper will discuss knowledge compilation and give an example of its use to simulate learning of initial programming skills. The rest of the paper will be devoted to discussing a recent discovery we have made about the knowledge compilation mechanism--that it can be used to produce learning effects that had been thought to be the result of independent learning mechanisms of discrimination and generalization (Anderson, Kline, and Beasley, 1979; Hayes-Roth and McDermott, 1976; Langley, 1983; Michalski and Stepp, 1983; Mitchell, 1978; Vere, 1975). Further I will argue that there is evidence in the human case that knowledge compilation is the process that underlies generalization and discrimination.

Introduction to Knowledge Compilation

The knowledge compilation mechanisms are defined with respect to a production system like ACT (Anderson, 1983a) which has a separate long-term declarative memory to represent facts and a production memory to represent procedures. The knowledge compilation mechanisms operate on the traces of production application to create new productions. Before getting into the details of a full example or the details of the implementation, it would be worthwhile to have a brief overview.

The knowledge compilation processes in ACT can be divided into two subprocesses. One, called *composition*, takes a sequence of productions that follow each other in solving a particular problem and collapses them into a single production that has the effect of the sequence. The idea of composition was first developed by Levis (1978). Composition speeds up the process considerably by creating new productions that embody the sequence of steps used in a particular problem domain. The second process, *proceduralization*, builds versions of the productions that no longer require the domain-specific declarative information to be retrieved into working memory so the information can be matched by the general interpretive productions. Rather, the essential products of these retrieval operations are built into the new productions.

The basic processes of compilation can be illustrated with respect to dialing telephone numbers. It has been noted (Anderson, 1976) that with a frequently dialed telephone number, one develops a special procedure for dialing it; sometimes declarative access to the number is lost, and the only access one has to the number is through a procedure for dialing it.

Consider the following two productions that might serve to dial a telephone number:

P1

IF the goal is to dial = telephone-number
and = digit is the first digit of = telephone-number
THEN dial = digit.

P2

IF the goal is to dial = telephone-number
and = digit1 has just been dialed
and = digit2 is after = digit1 in = telephone-number
THEN dial = digit2.

Composition creates "macroproductions" that do the operation of a pair of productions that occurred in sequence. Applied to the sequence of P1 followed by P2, composition would create:

function corresponding to LIST1. GRAPES associated this error with the failure to correctly specify the argument to CAR. On previous occasions BR had encountered the same error at the top level typing in commands like (CAR (A)) where the argument to CAR is to be taken literally rather than evaluated. Always in the past she had repaired these errors by *quoting* the argument. This is done by preceding the argument with a single quote, i.e. (CAR '(A)). We assume that both the subject and GRAPES have compiled from previous experience a rule that the way to repress this error is by using quote, which stops LISP from evaluating. After applying this rule the function becomes:

```
(DEFUN FIRST (LIST1)
  (CAR '(LIST1)))
```

When this new function is tried on an example, LISP returns the CAR of '(LIST1) which is LIST1--rather than the first element of the value of LIST1. It is at this point that the tutor intervenes and reminds the subject of how she would solve the problem at the top level. In this context the subject would have used (CAR LIST1) rather than (CAR (LIST1)) or (CAR '(LIST1)). We simulated this intervention in GRAPES by refocusing it on the code-argument goal in Figure 1, and putting (CAR LIST1) as a top-level example in working memory. Then GRAPES, as the subject, maps this code to its current function definition and comes up with the correct code:

```
(DEFUN FIRST (LIST1)
  (CAR LIST1))
```

Knowledge Compilation

After finally solving this problem, The knowledge compilation mechanisms formed two productions which aided its solution of the second problem. These productions summarized much of the problem solving that took place.

C1

```
IF the goal is to write a function of one variable
THEN write (DEFUN function (variable)
  and set as a subgoal to code the relation calculated by
    this function
  and then write ).
```

C2

```
IF the goal is to code an argument
  and that argument corresponds to a variable of the function
THEN write the variable name.
```

We have encircled in Figure 1 the portions of the goal tree that are summarized by each of these productions. The first production captures the top level syntax of a function while the second summarized the search involved in finding out how to specify an argument to a function. With these productions, GRAPES was able to write a second function much easier, as was the subject. This function, called SECOND, was to return the second element of a list.

Conclusions

There are a number of conclusions that we draw from this example. The first is the importance of structural analogy to bridging the gap between current knowledge and the needed behavior. We see two sources for the structure from which the analogy is being made.

One is templates provided in the text and worked-out problems. The other is structures that the subject can generate--for instance, the subject generated (CAR LIST1) as top-level solution and then used this in her function definition.

A second conclusion concerns the hierarchical structure of the problem-solving episode as illustrated in Figure 1. Note also that the goals in this tree are expanded depth-first and left-to-right. Jeffries, Turner, Atwood and Polson (1981) also note this hierarchical structure in the programming behavior of experts--although their subjects use breadth-first expansion in correspondence with the edicts of structured programming.

The third conclusion is the importance of knowledge compilation in extracting new production rules from an example problem. These rules streamline the solution of later problems. As the protocol shows, the learning can be on the basis of a single example. It needs to be stressed that the lessons of this example "stuck" which is to say, BR did not have, on later days, the same difficulty with the basic syntax of function definition nor argument specification. It should also be stressed that compilation depends critically on the structure of goal trees. That is, the structure of the goal tree identifies what parts of the problem-solving episode belong together and can be collapsed into a single rule.

In these three features--structural analogy, hierarchical goal trees, and knowledge compilation--we have one complete solution to the issue of how the subject is able to make the transition to a new cognitive behavior.

Further Discussion of Compilation

It would be useful to consider some further examples of proceduralization and composition, the two subprocess of compilation, for the domain of LISP programming.

Proceduralization

Proceduralization can be illustrated in its pure form by the following example: In GRAPES there is a production that will retrieve function definitions from long-term memory and apply them:

```
IF the goal is to code a relation on an argument
  and there is a LISP function that codes this relation
THEN use this function with the argument
  and set as a subgoal to code the argument
```

In this production, *relation* and *function* are variables which allow the production to match different data. The second line of the condition might match, for instance, "CAR codes the first member of a list." If this rule is proceduralized to eliminate the retrieval of the CAR definition, it becomes

```
IF the goal is to code the first member of a list
THEN use CAR of the list
  and set as a subgoal to code the list
```

This is achieved by deleting the second clause in the first production that required memory retrieval. Rather, the rest of the production is made specific to the relation first element and the function CAR. Now a production has been created which can directly recognize the

ones to do structural analogy and ones that could use the LISP functions CAR and CDR at the top level. The first type of productions represent a general prior skill evoked in many contexts (for instance, in filling out income tax forms). The second type was acquired from work with earlier chapters in Winston and Horn.

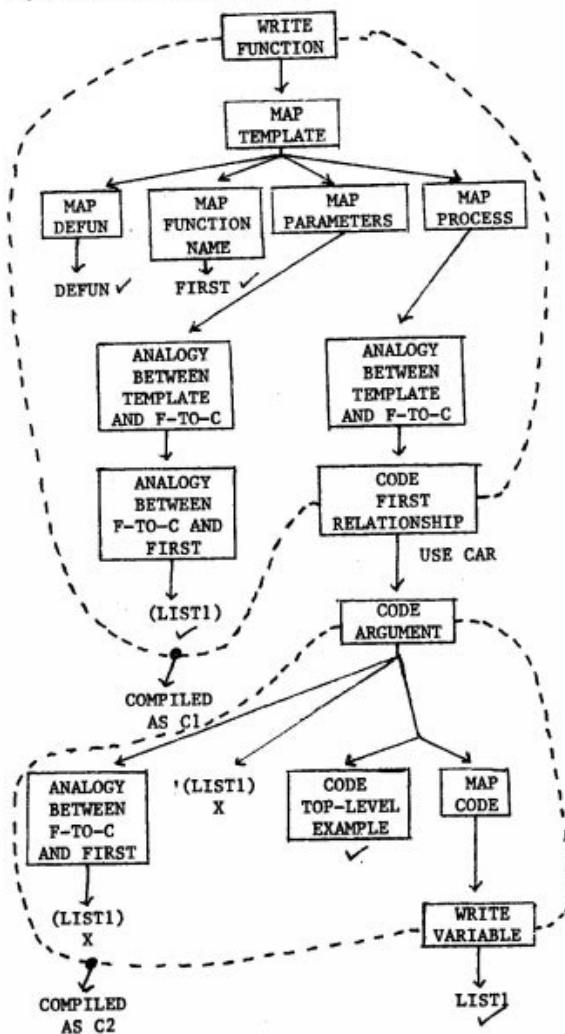


Figure 1.

Figure 1 illustrates the goal tree generated in simulating this example. Each box in Figure 1 represents a goal and each arrow emanating from the box represents a GRAPES production trying to achieve the goal. If the production has subgoals, it is connected to goal boxes below. The simulation starts with the goal of writing the function and chooses as its method to use the template for function definition as model. We refer to this as *mapping* the template. The first subgoals that GRAPES processed in Figure 1 were concerned with mapping DEFUN and <function name> in the template. Productions responding to these goals wrote out "(DEFUN FIRST" without difficulty.

Like our subject, GRAPES was not able to directly write out the parameter part of the function template because GRAPES did not know what a "parameter" was. In cases like this, GRAPES' analogy productions will resort to a concrete example. The concrete example retrieved by GRAPES is the definition of F-TO-C given earlier. The subject reviewed two examples in Winston and Horn—F-TO-C and EXCHANGE. In both cases she noted the single argument which was the parameter. GRAPES solved the analogy: X is to F-TO-C as the parameter list (i.e., (<parameter 1><parameter 2>...<parameter n>)) is to the abstract template, and retrieved (TEMP) as the value for X. Thus, it decided (TEMP) was serving the parameter role in F-TO-C. Then it solved the analogy X is to FIRST as (TEMP) is to F-TO-C and came up with the answer (LIST1) which it put into the function definition. That is, GRAPES decided (LIST1) served the comparable role in the function it was defining as (TEMP) was serving in F-TO-C.

Then GRAPES turned to the process description. Being unable to directly interpret what is meant by <process description>, it looked to its concrete example F-TO-C and saw that LISP code filled this slot which performed the function operations. In analogy, GRAPES set its goal to write code that would perform the operations required by FIRST. The subject at this point looked to a different example, INCREASE, for the same purpose of analogy. A production for using CAR at the top level applied next in GRAPES, but there is no production to specify how to write the argument to the function to CAR in this context. GRAPES and the subject know that CAR will operate on LIST1, but they do not know the syntax for specifying it. GRAPES again turns to its concrete example, F-TO-C, and solves the analogy (CAR ARG) is to (QUOTIENT X) and retrieves (DIFFERENCE TEMP 32) which is the first argument to QUOTIENT. It then solves the analogy problem of what it must do to LIST1 to make it like (DIFFERENCE TEMP 32) and decides it should embed LIST1 in parentheses. Similar to GRAPES, the subject looks at a function EXCHANGE which has the same argument structure as F-TO-C. We assume she makes the same erroneous analogy because she writes (LIST1). The first argument to QUOTIENT in F-TO-C is embedded in parentheses (as is the first argument to LIST in EXCHANGE) because a function, DIFFERENCE, must be called to calculate the argument for QUOTIENT whereas no function call is required for the argument to CAR in the FIRST example. The function definition at this point by both subject and GRAPES is:

```
(DEFUN FIRST (LIST1)
  (CAR (LIST1)))
```

There are two things to note at this point. First, the subject has read, in the text, information that would have enabled her to know not to embed LIST1 in parentheses, but this has no impact on her behavior. Second, on previous occasions she had correctly specified variable arguments when evaluating functions at the top-level. Eventually, the tutor used this second fact, that the subject could do it correctly at the top level, to guide her to a correct solution. Both of these observations illustrate the relative isolation of knowledge. That is, knowledge studied or used in one context is not available in another context.

When the subject tries her function definition, an error message is generated—"LIST1: undefined function object". GRAPES received the same error message when it tried out the same function definition. The error occurred because LISP treats the first thing inside a parenthesized expression to be evaluated as a function and there is no

S2

IF Gail is rich
and Gail is ugly
and Gail is stupid
and Gail is of medium height
THEN Gail is in club 1.

Then the following production would be formed as the generalization of the two:

G1

IF =person is rich
and =person is ugly
and =person is of medium height
THEN =person is in club 1.

where =person is a variable. The first productions can be viewed as specific observations and G1 is a generalization formed from these observations. Note G1 is formed both by deleting condition clauses and by replacing constants by variables. These are the two transformations that occur when one moves to generalizations.

The interesting observation is that compilation results in clause deletion and replacement of constants with variables. Compilation deletes clauses associated with omitted goals and with planning. On the other hand variables can remain from the planning process in the compiled productions. Thus it appears we might be able to get the effect of generalization through compilation.

Specifically, it appears that generalizations can be formed through the process of compiling analogies. The basic framework is as follows: The system has instances committed to memory as declarative facts. When a new instance is encountered, the system compares it to a memorized instance and uses the structure of the memorized instance to guide response. Then compiling this compare-and-respond behavior produces a production which is a generalization from the two instances. This is basically what happened in the case of forming C1 and C2 from Figure 1. I would like to go through a simpler example here that will make this process more transparent. Then, I will discuss the psychological evidence that indicates this is the correct conception of generalization in humans.

Consider a very simple production set that will classify new instances according to their similarity to studied instances:

P1

IF the goal is to classify =object
and =reference has been studied
THEN set as subgoals to compare =object to =reference
and to determine if =object is in the same category,
as =reference.

P2

IF the goal is to compare =object to =reference
THEN initialize the measure of overlap
and set as subgoals to match all the features of =object
to =reference.

P3

IF the goal is to match =feature to =reference
and =reference has =feature
THEN increment the measure of overlap.

P4

IF the goal is to match =feature to =reference
and =reference does not have =feature
THEN decrement the measure of overlap.

P5

IF the goal is to determine if =object is in the same category
as =reference
and the measure of overlap is above threshold
and =reference is in =category
THEN =object is in =category.

P6

IF the goal is to determine if =object is in the same category
as =reference
and the measure of overlap is below threshold
THEN POP failure.

This production set will keep comparing the to-be-classified object to candidate references from memory until a high similarity reference is found. Then it will categorize the object as in the same category as the high similarity reference.

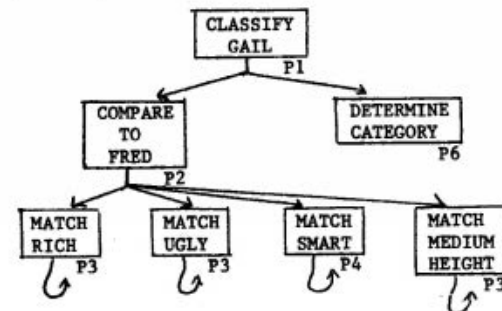


Figure 2.

Suppose the system has committed to memory that Fred is in category 1 and he was rich, ugly, smart, and of medium height. Then the system is asked to categorize Gail who is rich, ugly, stupid, and of medium height. Figure 2 illustrates the goal tree generated by GRAPES in applying this production set to the classification problem. It found matches on three features and mismatch on one--which was sufficient to exceed threshold. The compilation process regarded all the similarity comparison as planning goals and compiled all of this out and formed the following single production:

IF the goal is to classify =object
and =object is rich
and =object is ugly
and =object is of medium height
THEN =object is in club 1.

application of CAR. This will result in a reduction in the amount of long-term memory information that needs to be maintained in working memory.

Composition

As an example of pure composition, suppose one wanted to add the first member of List1 to List2. Then the following two operators would apply in sequence:

```
IF the goal is to add an element to a list
THEN CONS the element to the list
    and set as subgoals to code the element
    and to code the list
```

```
IF the goal is code the first member of a list
THEN use CAR of the list
    and set as a subgoal to code the list
```

The first rule above would apply binding *an element* to "the first member of List1" and *a list* to "List2". The second production would apply binding *a list* to "List1". A simple case of composition would involve combining these two productions together to produce

C4

```
IF the goal is to add the first member of one list to another list
THEN CONS the CAR of the first list to the second list
    and set as subgoals to code the first list
    and to code the second list
```

Such compositions collapse repeated sequences of coding operations to create macro-operators. The result is a speed-up in coding. A major issue concerns what productions to compose together. The above example is a fairly simple case of collapsing two levels of a goal tree into one. However, in some cases, such as when Figure 1 was collapsed into two productions, many productions can be collapsed. GRAPES determines what productions to collapse by inspecting the goal tree. There are two types of goals for purposes of composition. There are *inherent goals* and there are *planning goals*. Inherent goals are intrinsic parts of the problem domain. For current purposes inherent goals are all variants of writing code. The important feature of inherent goals is that in achieving them one achieves part of the original task. On the other hand, planning goals produce results that are used to guide solution of the original problem but the results themselves are not part of the original problem. In Figure 1 the inherent goals are "CODE THE FUNCTION", "CODE THE RELATIONSHIP", "CODE THE ARGUMENT", and "WRITE THE VARIABLE"; all the rest are planning goals.

Composition collapses productions in one of two ways. One possibility is to eliminate the planning goals that are intermediate between two inherent goals. This is what happened when C1, given earlier, was formed for Figure 1. Composition formed a rule that went from the goal of "CODE THE FUNCTION" to the goal of "CODE THE RELATIONSHIP". In doing this it compiled out the planning process and simply left in the products of that planning. The same process underlies the formation of C2 from the goal tree in Figure 1.

The second possibility for composition is illustrated in the case of C4 given earlier under the discussion of pure composition. Here it starts with the goal of adding the first element of one list to another, skips over the intermediate goal of coding the first element of the list, and sets the goals of coding the two lists. In so doing this it is basically creating macro operators somewhat similar to STRIPS (Fikes and Nilsson, 1971).

This learning scheme requires that the learner be able to identify what subgoals were essential to the problem solutions and which are only intermediate to the final solution.

It needs to be emphasized that neither proceduralization nor composition eliminate the original production rules from which they were built. Rather the new compiled rules just serve as additional supplemental rules to produce better performance in certain circumstances.

The effect of the knowledge compilation process is to create a set of productions that mirror the structure of LISP. They may explicitly involve LISP functions like CAR or COND or LISP programming techniques like tail-recursion. These productions will preserve the inherent goals which are specific to LISP and will delete the planning goals involved in domain-general processes like structural analogy. Thus representative productions become (see Anderson, Sauters and Farrell, 1982):

```
IF the goal is to code the second member of a list
THEN use CADDR to set a subgoal
    to code the list.
```

```
IF the goal is to obtain all the elements which have
    a relation to any member of a list
THEN use MAPCONC and set as subgoals
    To code a function that will return all the elements that
        have relation to the argument.
    To code the list.
```

Generalization

Generalization is another learning mechanism which can be defined on productions. As it is typically formulated (eg., Anderson, Kline, and Beasley, 1980), it takes a pair of productions and generates what is called the *maximal common generalization*. This is the most specific production that will apply in all the circumstances of either original production and have the same effect as the original productions. So, for a simple example, if we had the following pair of productions:

S1

```
IF Fred is rich
    and Fred is ugly
    and Fred is smart
    and Fred is of medium height
THEN Fred is in club 1.
```

The basic scheme for getting discrimination through compilation is to have the system consciously and deliberately go through the steps involved in forming a discrimination. This means doing discrimination by the computation of productions rather than having it be an automatic process sitting above the production system. Thus, a sequence of productions will compute a discrimination. Compiling that sequence will result in a discriminated production. As we will see, doing this is not as technically straightforward as the generalization case. This is because it would be necessary to inspect the contents of productions to perfectly mimic the automatic discrimination process whereas this is not necessary to mimic generalization. In the ACT theory one production cannot inspect the contents of another. Still we can get nearly the effect of automatic discrimination in a fairly plausible way.

As noted earlier, there are multiple different ways of forming discriminations. This section will focus on the action discrimination as just sketched out. However, I think this general scheme will extend to other types of generalization. This scheme involves having the productions compute the discrimination and then compiling that computation. After illustrating compilation of an action discrimination, I will consider the evidence that this view of discrimination is psychologically correct.

This example will be developed within the framework of productions P1-P6 given earlier for classifying objects by analogy. The learning situation I have in mind is one in which the subject makes a classification of an object, receives feedback, and if the classification is incorrect the subject goes about trying to find a discriminating feature. We will assume the subject has compiled the following too-general production:

G2*
 IF the goal is to classify =object
 and =object is rich
 and =object is of medium height
 THEN =object is in club 1.

This is just a variant of G2 given earlier but now set up for the current framework. As earlier we will assume it misapplies to classify David who is smart, good-looking, rich, and medium height as in club 1. An error is detected and a goal is set to reclassify David.

The following three productions are relevant to the correction of a misclassification:

P7
 IF the goal is to classify =object
 and =object is in =category1
 but =object was classified as in =category2
 and =reference is in =category2
 THEN remove the classification of =object
 and set as subgoals to find a difference between =object
 and =reference
 and then to reclassify =object
 and then to correct the classification of =object.

P8
 IF the goal is to find a difference between =object
 and =reference
 and =object is in =category1
 and =object has =feature1 on =dimension
 and =reference has =feature2 on =dimension
 and =feature1 is different from =feature2
 THEN conclude =object is in =category1 because of =feature1.

P9
 IF the goal is to correct the classification of =object
 and =object has been classified as in =category2
 but =object is in =category1 because of =feature1
 THEN change the classified category of =object to =category1.

Production P7 will apply to correct the situation. It selects some instance that was in the incorrect category. Suppose again it selects Fred who is rich, ugly, smart, and of medium height. This is an instance the overgeneral production G2* would have correctly classified. There is no guarantee that the instance selected by P7 will be one that fits the overgeneral production, because P7 cannot inspect G2*. However, the spreading activation retrieval mechanisms in ACT (see Anderson, 1983) would tend to select an instance that overlaps highly with the current instance and hence is matched by the rule that classified the current instance. P7 sets goals to find a difference between David and Fred and then to reclassify David.

Production P8 will note that David is good-looking while Fred is ugly and so enter into memory the fact that "David is in club 2 because he is good-looking". Then the system will return to the goal of classifying David once again. Again G2* would apply to misclassify David as in club 1, but now production P9 can apply to correct the classification of David. P7 followed by P8 followed by G2* followed by P9 constitute a goal tree that can be composed together and proceduralized. The result will be the following production:

D1*
 IF the goal is to classify =object
 and =object is rich
 and =object is of medium height
 and =object is good-looking
 THEN =object is in club 2.

This is essentially D1 formed earlier by the automatic discrimination mechanism.

There are two essential features that distinguish this means to discrimination from the automatic discrimination of the current ACT theory. First, it requires that the learner initially make a classification of the object and then correct that classification if in error. Second, it requires that the learner make a conscious hypothesis about what distinguishes the current instance from prior instances that were in the hypothesized category. Both of these features have been confirmed in two series of experiments by Lewis and Anderson (in preparation). One series of experiments involved rules for proving triangles congruent in geometry while the other series involved rules for travelling through a maze. In both series subjects were given overgeneral rules to start with and had to discriminate these rules.

This production is essentially identical to the generalization G1 given earlier. Thus, compiling the process of make an analogy will result in a generalization.

What has been described is a procedure for getting the same effect as generalization. What reason is there to believe that it is the process underlying generalization rather than the more direct generalization mechanism? First, it is more plausible within the ACT production system framework. To produce a generalization in ACT it is necessary to start out with very specific productions like S1 and S2 given earlier. This is because the generalization mechanism only works on productions. These highly specific productions are patently non-intuitive and moreover the ACT theory would claim it is not possible to directly form such productions. So, an ACT theory that produces generalization through compilation is more plausible than one that uses the direct generalization mechanism.

Second, independent of the ACT framework, the evidence is against an automatic generalization mechanism that has no strategic component. Both Elio and Anderson(1983) and Kline(1983) have shown that the generalizations a subjects emerge with will depend on both their problem-solving set and the order of the examples. Thus, it does not seem that humans extract always and only the maximal common generalizations. The generalizations they do extract are determined by what they are looking for. This is better modelled by a system like the preceding that leaves similarity detection at the strategic level.

Third, notice that the system can classify on the basis of a single studied instance. A generalization scheme requires at least two instances to form a generalization before novel instances can be classified. Elio & Anderson (1981) showed that subjects can classify novel instances on the basis of similarity to a single studied instance when there is no possibility of generalization. This is consistent with the current conception. In addition, Elio and Anderson showed that subjects could better classify a novel instance when there was a relevant generalization that could be formed from two studied instances--even when overall similarity of studied to novel instances was kept constant. This shows that generalizations also do occur and all classification is not a matter of analogy--again consistent with the current scheme.

Also, experiments have been done where subjects can classify novel instances after studying instances unaware of their classification structure(Brooks, 1978). For instance, subjects learn to associate animals and cities to stimuli in a simple paired-associate experiment. After doing the paired-associate task, they are told that all the stimuli associated to new-world animals and cities form one category and all stimuli associated to old-world responses form another category. Subjects were unaware of the old-world versus new-world dimension during study. They can now reliably classify new instances as to whether they are old or new world stimuli. Since subjects were not aware of the classification structure at study there was no opportunity to form generalizations. The classification must be on the basis of comparing specific test stimuli to specific study stimuli.

Carbonell(1983) provides an elaborate discussion of how analogy might be used to guide problem-solving which goes beyond the discussion here. He also speculated on how generalizations might emerge from the analogy process. He proposed that the subject would store analogical solutions and generalize from these in just the way

typical generalization from examples works. This discussion shows that generalizations can emerge as a byproduct of the analogy process without a separate generalization phase.

Discrimination

Less work has been done on discrimination in the knowledge acquisition literature and there is less consensus about how to do discrimination. However, the general situation for discrimination is one in which there is an overly general rule and there is a need to restrict its range. I personally doubt that there is a single way to form discriminations and this probably accounts for the lack of consensus in the literature. I certainly do not think there is a uniform discrimination process in the human. However, for purposes of discussion I will focus on a kind of discrimination called *action discrimination* in the ACT theory(Anderson, 1982, 1983). Again for clarity I will illustrate it with respect to simple classifications but Anderson(1982) can be consulted for more complex applications.

Suppose we have the following general production:

G2

IF = person is rich
= person is of medium height
THEN = person is in club 1.

Suppose this rule is applied to David who is smart, good-looking, rich, and of medium height. It would classify David as being in club 1. However, suppose it is subsequently learned that David is in club 2. Thus, an error has been made and the system sets out to form a discrimination that will correct this error.

In forming a discrimination the system looks for some past instance where the rule correctly applied. Suppose, in this case, Fred is retrieved as someone correctly classified by the rule who is smart, ugly, rich, and of medium height. The discrimination mechanism looks for some feature which was true of the successful instance but was not true of the unsuccessful instance. In this case, the only difference is that David is good-looking while Fred is ugly. Therefore, a new production is formed that has this added feature:

D1

IF = person is rich
and = person is of medium height
and = person is good-looking
THEN = person is in club 2.

This is called an action discrimination because it has a different action than the production(G2) from which it was formed. D1 does not replace G2 but because of the specificity ordering of productions D1 will take precedence over G2 whenever both match. G2 will only apply when the person is not good-looking. Note in forming a action production the system both adds to the condition of a production and changes the action.

The interesting thing to note is that compilation can have the effect of adding to the condition of a production and changing the action. When we compose two productions, P1 followed by P2, we can add to the conditions of P1 some of the conditions of P2 and we can change the action of P1 to the action of P2. Thus, it appears that we might be able to get the effect of discrimination through compilation.

- Neves, D.M. and Anderson, J.R. Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.) Cognitive Skills and their Acquisition. Hillsdale, N.J.: Erlbaum, 1981.
- Sauers, R. and Farrell, R. GRAPES User's Manual. Technical Report ONR-82-3, 1983.
- Vere, S.A. Induction of concepts in the predicate calculus. Proceedings of the Fourth International Joint Conference of Artificial Intelligence, 1975, 281-287.
- Winston, P.H. and Horn, B.K.P. LISP. Reading, MA: Addison-Wesley, 1981.

Subjects who passively studied instances failed to learn the discriminating features. Subjects only learned when they made active hypotheses about the correct rule which could then be disconfirmed. Second, the only subjects to learn were those who had some conscious awareness of what the discriminating features were. Thus, we found that subjects only discriminated when both conditions were satisfied for the current scheme of discriminating through compilation.

General Conclusions

There is ample evidence for something close to knowledge compilation as a basic process of human learning (Anderson, 1982; Anderson, Farrell, and Sauer, 1983; Neves and Anderson, 1981). The evidence had always been rather scarce for the ACT mechanisms of generalization and discrimination and more recently we have been gathering some rather negative evidence. However, it seemed clear that human learning required generalization and discrimination in some form even if not the ACT form. It now appears that the ACT knowledge compilation mechanisms can produce discrimination and generalization in a more plausible form.

It is something of an embarrassment that we worked so long with the ACT mechanisms before realizing how the knowledge compilation mechanisms could be recruited to provide the effect of generalization and discrimination. This is because we thought of knowledge compilation as simply making more efficient existing paths of processing rather than enabling novel paths of processing. To get novel behavior it seemed that we needed inductive mechanisms of learning such as generalization and discrimination. What we failed to recognize was that if we compiled the results of acting on the basis of similarity detection and difference detection we could produce productions that enabled novel behavior.

References

- Anderson, J.R. Language, Memory, and Thought. Hillsdale, NJ: Erlbaum, 1976.
- Anderson, J.R. Acquisition of cognitive skill. Psychological Review, 1982, 89, 369-406.
- Anderson, J.R. The Architecture of Cognition. Harvard University Press, 1983a.
- Anderson, J.R. Acquisition of proof skills in geometry. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), Machine Learning. Palo Alto, CA: Tioga press, 1983b.
- Anderson, J.R., Sauer, R., and Farrell, R. Learning to Plan in LISP. ONR Technical Report 82-1. Carnegie-Mellon University, Pittsburgh, PA., 1982.
- Anderson, J.R., Kline, P.J., and Beasley, C.M. A general learning theory and its application to schema abstraction. In G.H. Bower (Ed.) The Psychology of Learning and Motivation, Vol 13. New York: Academic Press, 1979.
- Anderson, J.R., Kline, P.J., and Beasley, C.M. Complex Learning Processes. In R.E. Snow, P.A. Frederico, and W.E. Montague (Eds.) Aptitude, Learning, and Instruction, Vol II. Hillsdale, NJ: Erlbaum, 1980.
- Brooks, L. Nonanalytic concept formation and memory for instances. In E. Rosch and B.B. Lloyd (Eds.), Cognition and Categorization. Hillsdale, NJ: Erlbaum, 1978.
- Carbonell, J.G. Learning from analogy: Formulating and generalizing plans from past experience. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.) Machine Learning. Palo Alto, CA: Tioga Press, 1983.
- Elio, R. and Anderson, J.R. The effects of category generalization and instance similarity on schema abstraction. Journal of Experimental Psychology: Human Learning and Memory, 1981, 7, 397-417.
- Elio, R. and Anderson, J.R. The effects of information order and learning mode on schema abstraction. Memory and Cognition, 1983, in press.
- Fikes, R.E. and Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 1971, 2, 189-208.
- Hayes-Roth, F. and McDermott, J. Learning structured patterns from examples. Proceedings of the Third International Joint Conference on Pattern Recognition, 1976, 419-423.
- Jeffries, R., Turner, A.A., Polson, P.G., and Atwood, M.E. The processes involved in designing software. In J.R. Anderson (Ed.) Cognitive Skills and their Acquisition. Hillsdale, N.J.: Erlbaum, 1981.
- Langley, P. A general theory of discrimination learning. In D. Klahr, P. Langley, and R.T. Neches (Eds.) Self-modifying Production System Models of Learning and Development. Cambridge, Mass.: Bradford Books, 1983.
- Lewis, C.H. Production System Models of Practice Effects. Ph. D. dissertation, University of Michigan, 1978.
- Lewis, M. and Anderson, J.R. The role of feedback in discriminating problem-solving operators. In preparation, 1983.
- Michalski, R.S. Learning from observation: Conceptual clustering. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.) Machine Learning. Palo Alto: Tioga Press, 1983.
- Mitchell, T.M. Version Spaces: An approach to concept learning. Ph.D. dissertation, Stanford University, 1978.

both cognitive psychology and machine learning. There are certain formal similarities in models across these domains that suggest some constraints on concept naturalness.

1.3.2 Independent attribute models

Posner and Keele (1968) proposed that people form an impression of the central tendency of a category as a result of experience with exemplars and that categorical judgments come to be based on this central tendency or prototype. An ostrich, for example, would be judged to be a bird and not a mammal because it is more similar to the bird prototype than to the mammal prototype.

What does prototype theory imply about the structure of categories? The main constraint is that the process has to work in the sense that all category members will be accepted and all nonmembers rejected. If it happened that canaries were mammals and not birds, a prototype process would not work because canaries have many characteristic properties of birds and few characteristic properties of mammals.

There are corresponding models in the literature on machine pattern recognition (e.g., Nilsson, 1965). Of the many elegant and powerful pattern recognition algorithms that have been proposed (see, for example, Chen, 1973), particular interest attaches to a special case of discriminant analysis, namely, linear discriminant functions. If A is a pattern having n properties $A = (a_1, a_2, \dots, a_n)$, then $g(A) = w_1 a_1 + w_2 a_2 + \dots + w_n a_n + w_{n+1}$ is a linear discriminant function where $w_1, w_2, \dots, w_n$ are property weights (and w_{n+1} is an additional constant included for computational convenience) associated with the set of properties.

If exemplars (each having n properties) from two different categories are to be classified, then one can attempt to find a set of weights on these properties such that the linear discriminant function yields higher values to all instances of one class than to those of the other. If such a function can be found, then the categories can be partitioned on the basis of the discriminant function and the categories are said to be linearly separable. This constraint can also be described in geometric terms. Two categories are linearly separable when a boundary exists between the two sets of instances, this boundary being a straight line if the instances are described in two-space, a plane if the instances are described in three-space, and so on.

There is a formal similarity between linear discriminant functions and prototype models. One way of thinking about classifying stimuli on the basis of similarity to prototype is that it involves a summing of evidence against a criterion. If an instance has a sufficient number of bird properties then it will be classified as a bird. Whether one thinks of the process as a global similarity judgment or a summing of evidence, for a prototype process to work perfectly the categories must be linearly separable. Whichever formal language one adopts, models based directly

or indirectly on a weighted function of properties, considered independently, imply that linearly separable categories should be easier to master than nonlinearly separable categories. Because this class of models assumes that properties are processed independently, they will be referred to as independent attribute models.

1.3.3 Relational coding models

There is a set of classification models that do not imply that linear separability is an important constraint on categories. These alternative models have in common the assumption that classification involves in one way or another combinations or interactions of properties or attributes (e.g., Anderson, Kline, & Beasley, 1979; Hayes-Roth & Hayes-Roth, 1977; Medin & Schaffer, 1978; Michalski, 1980). We refer to this class of models as relational coding models.

1.3.4 Contrasts between independent attribute and relational coding models

Since it is difficult to specify the attributes comprising natural concepts, researchers interested in developing classification theories have often turned to artificially constructed categories. The structure of these categories is designed to mirror some property of interest, allowing one to examine learning without the additional burden of having to deduce the underlying attributes. This is the strategy that we have adopted to evaluate the importance of linear separability in human categorization. Specifically, two categorization tasks are set up to be similar in major respects, except in one case the categories are linearly separable and in the other case they are not. Independent attribute models imply that the task involving linearly separable categories should be easier to learn than the task not conforming to this constraint.

II. EVIDENCE BEARING ON LINEAR SEPARABILITY

2.1. General experimental design

Categories that either are or are not linearly separable are shown in Figure 1 (taken from Experiment 1 of Medin & Schwanenflugel, 1981). In this example, stimuli are comprised of values on four component dimensions, described in terms of a binary notation. The value 1 represents the typical or characteristic value for members of category A and the value 0 is typical for category B members. No defining features exist to partition categories A and B, but the linearly separable categories in the top half of Figure 1 can be separated on the basis of characteristic features. One simple way to see this is to focus on the first three dimensions comprising category members. Every category A member and no category B member has two of the three characteristic values (value 1) for category A. Therefore, every exemplar could be unambiguously classified by checking to see if a majority of its values on the first three dimensions were consistent with category A or with category B.

The categories shown in the bottom of Figure 1 have the same overall distribution of values

LINEAR SEPARABILITY AND CONCEPT NATURALNESS

Douglas L. Medin
Psychology Department
University of Illinois
Champaign, IL 61801

ABSTRACT

Many theories of human concept learning and some models for machine pattern recognition imply that linearly separable categories should be easier to master than categories which are not linearly separable. When categories are linearly separable, classification can be thought of as analogous to a summing of evidence against some criterion. An accumulating body of evidence suggests that such a summing of probabilistic evidence is not very natural for learners and that linear separability is not an important constraint on human classification performance. When learners perceive the various attributes or sources of evidence to be related to some superordinate theory, however, summing of evidence becomes more natural and linearly separable categories are easier to master than nonlinearly separable categories. These results are consistent with the idea that concept naturalness cannot be described solely in terms of abstract structural properties (such as the presence or absence of linear separability) and that people's theories about categories may determine which operations or descriptors are natural.

Key terms: Ill-defined categories, classification learning, linear discriminant functions, prototype models, machine pattern recognition, concept naturalness.

I. INTRODUCTION

1.1. Main focus

Classification is a fundamental cognitive task. Our entire system of naming objects and events, talking about them and interacting with them presupposes the ability to group experiences into appropriate categories. Since experiences might be partitioned in a limitless variety of ways, an important question is why we have the categories we have and not others. That is, what structural principles or constraints characterize natural concepts? We will assume that natural concepts should be easy for people to learn although it would be incorrect to make the converse assumption that easily learned concepts necessarily are natural. As will be seen, alternative theories imply differing constraints on category structure.

1.2 Concept structure

There is general agreement that natural categories such as bird, fruit, furniture, and tools are structured so that exemplars within a category are more similar to one another than to exemplars from alternative categories. There is considerable disagreement, however, concerning the rigidity of this structure.

1.2.1 The classical view

One view is that all natural concepts are characterized by simple sets of defining properties or attributes that are singly necessary and jointly sufficient to determine category membership (e.g., Katz & Postal, 1964). A candidate exemplar either does or does not possess these defining properties and in this sense classical view concepts are well-defined.

1.2.2 The probabilistic view

For a variety of reasons the classical view of concepts has fallen into disfavor (see Smith & Medin, 1981, for a detailed review). The consensus now has it that most natural concepts are not well-defined but rather are based on relationships that are typical or only on the average true. Members of a category may vary in the number of characteristic properties they possess and correspondingly vary in the degree to which they are judged to be good examples (typical) of the concept (Rosch, 1973; Smith, Shoben & Rips, 1974; Mervis & Rosch, 1981). For example, robins have the characteristic properties of flying, building nests in trees, and singing and are rated to be better examples of the category bird than are ostriches who manifest none of these properties (McCloskey & Glucksberg, 1978). Not only is it the case that category membership judgments appear to be based on characteristic properties, in many cases laypeople and experts alike cannot specify the defining properties of concepts (e.g., furniture) that are required by the classical view. The idea that many natural concepts are ill-defined will be referred to as the probabilistic view.

1.3 Learning models for ill-defined categories

1.3.1 Models and constraints

If many natural categories do not have defining properties, how do people acquire and use them? Candidate models and theories exist in

Strategy might be to assume that this is a fixed constraint and that results that hold for one realization of abstract category structure will hold for all realizations of that same abstract structure. The apparent generality of our results supports this approach. An alternative strategy, which we have followed, is to consider the possibility that what is natural may interact with the particular knowledge structures or theories that might be brought to bear on a problem.

IV. THEORIES AND LINEAR SEPARABILITY

4.1. Theories and summing evidence

Our basic idea is that if learners perceive the various attributes or sources of evidence to be related to some superordinate theory, then the idea of summing evidence may become more natural. In terms of induction models based on descriptors such as Michalski's (1980) rule-guided system, the theory might suggest that a new descriptor based on a summing of attribute values be constructed and evaluated.

4.2. Additional linear separability experiments

4.2.1. Linear separability

The logic of our recent experiments is quite simple. Participants are tested on category structures that either are or are not linearly separable and by means of assignment of dimensional values to the abstract category structure or by instructions we either do or do not provide the basis for a theory about the categories to influence performance.

4.2.2. Experiments

In one recent experiment (conducted in collaboration with Tim Murphy) the stimulus properties were descriptions of behaviors of people. The people belonged to one of two categories which were given arbitrary names. The behaviors were relevant to the trait dimensions of talkative, honest, cooperative, or cautious. For the theory condition, the typical values for one category all corresponded to a single trait and the typical values for the other category corresponded to its polar opposite. For example, if the superordinate trait were talkative then the typical behaviors for one category would be things like "told a lot of funny stories at a party" and "went out of his way to talk to a neighbor," and the corresponding behaviors in the other category would be things like "didn't interrupt others when they were talking at the party" and "remained silent when others were gossiping about their neighbor." In the control condition each of the four behaviors associated with the exemplars was related to a different one of the four traits. In both conditions the abstract description or notation would be analogous to that shown in Figure 1 but in the theory condition the dimensions corresponded to locations in which the behaviors occurred (park, home, party, store) and in the control condition the dimensions corresponded to the four different traits. We expected that in the theory condition participants would

perceive the behaviors as relevant to a unifying trait and would find it natural to use a summation of typical values as a decision rule.

The results fit with this expectation. There was a significant interaction between category structure and theory versus control condition. Participants in the theory condition learned the linearly separable tasks more readily than the nonlinearly separable task (mean errors during learning 27.4 versus 38.1) whereas the opposite result obtained in the control condition (mean errors for linearly separable group = 49.9, for nonlinearly separable group = 38.9).

In a related followup experiment (conducted in collaboration with Gerald Dewey and Glenn Nakamura), the attributes were descriptions of objects. The description pairs were "made of metal" versus "made of rubber," "medium weight" versus "light weight," "flat surface" versus "irregular surface," and "easy to grasp" versus "difficult to grasp." The modal or typical values for one category were the first mentioned value of each pair and the modal properties for the other category were the opposite values. In the control condition standard instructions were used. In the theory condition an additional sentence was added: "It may help you to know that the objects in one of the categories might serve as reasonable substitutes for a hammer."

The results were straightforward. In the control condition the nonlinearly separable task was easier (mean errors 52.2 versus 57.6) but in the experimental condition the linearly separable task was easier (mean errors of 31.2 versus 43.0).

Finally, we have evidence that the theory and the attribute structure must be compatible if a summing of evidence is to be evoked. William Wattenmaker and I performed an experiment where the stimuli were properties of animals and participants either were or were not informed that the categories would map onto the categories predator and prey. We did not observe an interaction of category structure and instructions. We think the absence of an interaction derives from the fact that summing of evidence was not natural. For example, it's a good adaptation for prey to be armored or to live in a tree, but it might not have been natural for participants to think that both being armored and living in a tree is a still better adaptation.

V. SUMMARY AND IMPLICATIONS

The overall pattern of results provides a relatively clear but mixed picture. A preponderance of evidence suggests that linear separability is not a major constraint on human classification. This evidence is consistent with relational coding models and inconsistent with independent attribute models. When learners perceive the component attributes or sources of evidence to be related to a compatible superordinate theory, however, then summing of evidence becomes more natural and linearly separable categories are easier to learn than nonlinearly separable categories.

but are not linearly separable. The difficulty is that exemplar A₁ has more values characteristic of category B than of A. Only its value on the first dimension is typical for category A, and it would not help to give the first dimension a large weight because then exemplar B₂ would erroneously be categorized as an A. One can see that A₁ and A₂ are both placed in category A, and B₁ and B₂ are placed into category B. The categories are not linearly separable. Thus, if linear separability is as important in human classification as independent attribute models imply, then it should be easier for people to learn to categorize the stimuli in the top of Figure 1 than to master the task in the bottom of Figure 1.

LINEARLY SEPARABLE CATEGORIES

Category A					Category B				
Exemplar	Dimension				Exemplar	Dimension			
	D ₁	D ₂	D ₃	D ₄		D ₁	D ₂	D ₃	D ₄
A ₁	1	0	1	1	B ₁	1	0	0	1
A ₂	1	0	1	0	B ₂	0	0	1	0
A ₃	1	1	0	1	B ₃	0	1	0	0
A ₄	0	1	1	0	B ₄	0	0	0	1

CATEGORIES NOT LINEARLY SEPARABLE

Category A					Category B				
Exemplar	Dimension				Exemplar	Dimension			
	D ₁	D ₂	D ₃	D ₄		D ₁	D ₂	D ₃	D ₄
A ₁	1	0	0	0	B ₁	0	1	1	0
A ₂	0	1	1	1	B ₂	1	0	0	1
A ₃	1	1	1	0	B ₃	0	0	0	0
A ₄	1	0	1	1	B ₄	0	0	0	1

Figure 1. Abstract representation of the alternative categorization tasks. (Each task involves eight stimuli varying along four dimensions.)

2.2. Summary of results

The results are easy to summarize: there is no evidence that linear separability provides an important constraint on human categorization. Medin and Schwanenflugel (1981) reported a series of four studies which varied the type of stimulus material (geometric shapes or photographs of faces), instruction (focus on abstraction versus memorize exemplars), and category size (from four exemplars per category to an unlimited number of exemplars per category). In no case were linearly separable categories easier to learn than nonlinearly separable categories. In experiments where linear separability was pitted against number of cases of high between-category similarity (a factor which relational coding models imply is

important), the nonlinearly separable categories actually were easier to learn.

There is also much earlier work bearing on linear separability. In their classic monograph examining the relation between identification learning (learning exemplars) and classification, Shepard, Hovland, and Jenkins (1961) employed procedures where in some cases the categories were linearly separable and in other cases they were not. Since this contrast was not the object of their study, between-category similarity was not fully controlled and should have favored the linearly separable categories. Nonetheless, across variations in stimulus properties and in learning conditions Shepard et al. found no evidence that linearly separable categories were easier than nonlinearly separable categories.

More recently we have conducted additional studies using still further variations on stimuli, number of categories, and learning procedures. In no case has linear separability mattered. Overall, these findings are more compatible with relational coding models than with independent attribute models.

III. FURTHER CONSIDERATIONS

3.1. Typical values and summing evidence

The failure of linear separability to provide an important constraint on categorization might arise either because people have difficulty in learning the typical (or modal) values for a category or because they do not find it natural to employ a summing of typical values as a decision rule for categorization. Additional observations suggest that summing probabilistic evidence is the key factor.

First of all, when people are asked to provide that characteristic or typical values at the end of training they are quite accurate. Furthermore, procedures that should facilitate the learning of modal values do not differentially facilitate the learning of linearly separable versus nonlinearly separable categories. For example, Murphy (1983) gave participants two runs through the category exemplars prior to learning during which the exemplars were either blocked by category or randomly intermixed. Blocking facilitated learning but this facilitation was as large for nonlinearly separable categories as for linearly separable categories. Finally, we often ask participants at the end of the experiment how they learned the task, and only rarely do they mention anything like a summing of characteristic values. Instead, their descriptions more commonly are in terms of rules plus exceptions or a disjunction of conjunctions, descriptions that are more consistent with relational coding models.

3.2. Summing evidence and naturalness

The above observations suggest that participants in our experiments do not find it natural to sum up evidence to make categorizations, even when a two out of three rule would work perfectly.

How Can CHILD learn about Agreement? Explorations of CHILD's Syntactic Inadequacies

Mallory Selfridge
Department of EE and CS
University of Connecticut
Storrs, Ct. 06268

1.0 Introduction

The CHILD program has accounted for a large amount of data on child language learning, and manifests a developmental progression which is psychologically accurate [4,5,6,7]. However, its representation of language syntax is known to be inadequate in a number of ways. This is manifested in CHILD's inability to learn to understand and generate various syntactic constructions which children do learn. This paper is an initial attempt to address these issues. It first reviews the CHILD program, and then it describes those aspects of its syntactic representation upon which this performance rests. Third, it describes a subset of the problems which this representation introduces, and finally outlines possible directions for further development.

2.0 The CHILD Program

CHILD is a computer program which models the development of a child's language comprehension and generation abilities written in Franz LISP and currently running on a DEC VAX 11/780. It begins with world knowledge and language experiences similar to those received by children, and learns a subset of the word meaning and syntax which children learn. After learning, CHILD can correctly understand utterances which it previously misunderstood, and can generate English describing events it "observes."

CHILD's language comprehension process is a version of the CA program [1] which incorporates mechanisms derived from Wilks' [8] preference parsing. CHILD's analysis process combines Conceptual Dependency (CD) [2] word meanings to form a CD representing the meaning of the entire utterance. Understanding begins when the meanings of input words are placed in a short term memory. CHILD then retrieves semantic requirements associated with particular slots in the CD itself, and also syntactic features associated with those slots but specific to that particular word. It searches the short term memory for a word meaning which best satisfies those features, and fills the empty slot with that meaning. The syntactic features are formed from the positional predicates "precedes" and "follows." These relate the position of a candidate slot filler to either the word they were stored under, a filler of another slot in that

word's meaning, or a lexical function word. Each slot in the meaning of a word has a collection of features describing where in the input a filler is expected to be. In order to understand different voices, these features are organized into disjunctive "feature sets." Each set characterizes one order in which slot fillers appear. During understanding, feature set selection is performed by considering which set most successfully characterizes the input. In this paper, this representation will be called "concept order syntax"

CHILD learns syntax by acquiring syntactic features and build disjunctive feature sets. After having understood an utterance, CHILD examines a record of the input, and examines the meaning of every input word. It then examines every empty slot in each such meaning. It accesses the record of the input to find where in the input the filler for that slot occurred. It creates a description of this position using "precedes" and "follows." CHILD must then decide whether this description constitutes a new feature set or should be merged with an existing feature set. CHILD's strategy is based on a suggestion by Iba [8]. CHILD compares the features extracted from the current input with any existing feature sets: the position description is merged with a previous set only if one is a subset of the other. Otherwise, the description is learned as a new feature set.

3.0 Why is Concept Order Syntax Good?

The power of concept order syntax appears to derive first from its formulation as a set of independent features. This means that the syntactic decisions that have to be made during understanding and generation are not binary "something is either in the correct location or it isn't" decision. Rather, these decisions are more comparative: "this location is better than that." This results in smoother and more robust understanding and generating. A second advantage of independent features is that it makes learning easier. The features observed to be true in the input are merely added to those features already known to describe filler positions. This is far easier than trying to decide which word-class a word is in, for example. A third advantage of syntactic features is that it simplifies the understanding process enormously. This is because a large portion of understanding is manipulating semantic features. Having syntactic features means that syntactic processing can be handled by the same mechanisms that handle the semantic features, and at the same time. This is par-

The results imply that constraints provided by different types of category structure are not independent of the types of theories or knowledge structures brought to bear on them. So while the overall pattern of results favors relational coding models, these models will need to provide an account of how these knowledge structures modify the types of operators or descriptors that are natural.

VI. REFERENCES

- Anderson, J.R., Kline, P.J., & Beasley, C.M. "A general learning theory and its application to schema abstraction." In G.H. Bower (Ed.), The Psychology of Learning and Motivation (Vol. 12). New York: Academic Press, 1975.
- Chen, H.C. Statistical Pattern Recognition. Rochelle Park, NJ: Hayden Books, 1973.
- Hayes-Roth, B., & Hayes-Roth, F. "Concept learning and the recognition and classification of exemplars." Journal of Verbal Learning and Verbal Behavior, 1977, Vol. 16, pp. 321-338.
- Katz, J.J., & Postal, P.M. An Integrated Theory of Linguistic Descriptions. Cambridge, MA: MIT Press, 1964.
- McCloskey, M.E., & Glucksberg, S. "Natural categories: Well-defined or fuzzy sets?" Memory & Cognition, 1978, Vol. 6, pp. 462-472.
- Medin, D.L., & Schaffer, M.M. "Context theory of classification learning." Psychological Review, 1978, Vol. 85, pp. 207-238.
- Medin, D.L., & Schwanenflugel, P.L. "Linear separability in classification learning." Journal of Experimental Psychology: Human Learning and Memory, 1981, Vol. 7, pp. 355-368.
- Mervis, C.B., & Rosch, E. "Categorization of natural objects." In M.R. Rosenzweig & L.W. Potter (Eds.), Annual Review of Psychology, 1981, Vol. 32, pp. 89-116.
- Michalski, R.S. "Pattern recognition as a rule-guided inference system." IEEE Transactions on Pattern Analysis and Machine Intelligence, 1980, Vol. 2, pp. 349-361.
- Murphy, T.D. Stimulus Presentation Effects and the Processing of Ill-defined Categories. Unpublished M.A. Thesis, University of Illinois, 1983.
- Nilsson, N.J. Learning Machines. New York: McGraw-Hill, 1965.
- Posner, M.I., & Keele, S.W. "On the genesis of abstract ideas." Journal of Experimental Psychology, 1968, Vol. 77, pp. 353-363.
- Rosch, E. "On the internal structure of perceptual and semantic categories." In T.E. Moore (Ed.), Cognitive Development and the Acquisition of Language. New York: Academic Press, 1973.
- Shepard, R.N., Hovland, C.I., & Jenkins, H.M. "Learning and memorization of classifications." Psychological Monographs, 1961, Vol. 75, (13, Whole No. 517).
- Smith, E.E., & Medin, D.L. Categories and Concepts. Cambridge, MA: Harvard University Press, 1981.
- Smith, E.E., Shoben, E.J., & Rips, L.J. "Structure and process in semantic memory: A featural model for semantic decision." Psychological Review, 1974, Vol. 81, pp. 214-241.

Footnote

This research was supported in part by National Institute of Mental Health Grant MH32370 to Douglas L. Medin.

and passive versions of the same sentence are understood to mean exactly the same thing. A tighter coupling of syntax and meaning would allow passive sentences to have different meanings than their active versions. It would also allow the use of feature sets in question understanding, allowing discriminations between "Is that Tom?" from "That is Tom."

One possible technique to couple syntax and meaning more tightly is to interpose an intermediate layer of meaning representation between the feature set and the word meaning proper. In such a scheme, when a word is read in during understanding, its primary meaning would be placed on the c-list, as currently. Then understanding would proceed, and when feature set selection was completed the secondary meaning associated with that feature set would be added to the primary meaning. For example, the word "give" might have ATRANS as its primary meaning, including a FOCUS slot, and the secondary meaning associated with the feature sets would specify the filler of the FOCUS slot. Thus when a feature set was selected, the FOCUS slot would be filled with the name of the ATRANS slot being focussed. Thus the secondary meaning in this case would be the filler of a slot in the primary meaning.

The second possible extension concerns the problem of syntactic lexical specificity. Rather than associating syntactic knowledge with individual words, common syntactic feature sets should be stored "centrally," and each word should index those centrally stored feature sets. Since the centrally stored syntactic knowledge must contain exactly the same syntactic knowledge now stored directly under each word, central storage will have no less power than the current approach. It will, however, reduce storage space make learning syntax easier, and help explain the phenomena of syntactic generalization noted above.

The third possible extension concerns the problem of syntactic embedding. In many ways this is the most difficult problem of any here, and there is no clear answer. Indeed, this question is a central one for linguistics. While a definite extension cannot be described here, the direction in which such an extension lies can be described. One way characterization of the problem is that certain syntactic features are appropriate if referring to slot fillers at the top level of meaning representation. If the slot filler is not at the top level of meaning representation, on the other hand, then other syntactic features may be appropriate. This suggests that information should be included in a syntactic feature which refers to levels of meaning representation. Specifically, this information would specify that the feature is to be activated only if the meaning containing the slot which indexes the feature is the top-level meaning representation of the sentence. Other features will have similar information activating them only if the meaning containing the slot being filled is not the top level feature. While this approach is certainly very far from an ultimate explanation, it appears to represent an improvement over the current representation. It will enable the system's performance to improve, and it will provide additional experience with the problem of syntactic

embedding.

Finally, there is the question of agreement. How can concept order syntax be extended to describe knowledge of agreement? In "the dog eats" vs. "the dogs eat", the number of dogs seems to have a direct correlation with the presence of "s" on the end of "eat." This suggests a contingent component to agreement: "if there is an 's' following 'eat' then there is one dog, otherwise there are two." Since the number of dogs is clearly semantic, and the presence or absence of "s" clearly syntactic (unless a meaning can be assigned to it), then CHILD should be able to manipulate links between syntax and semantics of the form "if syntactic feature then semantic expectation about a certain slot filler." Such a formulation may allow CHILD to understand and generate correctly agreeing inputs; if so, such a successful representation would be the first step toward learning agreement.

6.0 Conclusions

The CHILD model of child language learning has demonstrated considerable performance, but has been limited by its representation of syntax. This paper is intended to acknowledge the existence of these limitations, and to outline current progress toward reducing some of them. Such progress will enable the CHILD program to learn more complex syntactic constructions, and would allow CHILD to account for additional data. It would also provide further experience with which to uncover additional inadequacies with both concept order syntax and with CHILD itself.

References

- [1] Birnbaum, L. and Selfridge, M. (1981). Conceptual Analysis of Natural Language, in *Inside Computer Understanding: Five Programs plus Miniatures*. Schank R. and Riesbeck C.K. (eds.), Lawrence Erlbaum Associates, Hillsdale, NJ.
- [2] Iba, G. (1979). Learning Disjunctive Concepts from Examples. M.I.T. A.I. Memo #548, M.I.T., Cambridge, Mass.
- [3] Schank, R. C., (1973). Identification of Conceptualizations Underlying Natural Language. R. C. Schank and E. M. Colby (eds.). *Computer Models of Thought and Language* W.H. Freeman and Co., San Francisco.
- [4] Selfridge, M. (1980). A Process Model of Language Acquisition. Computer Science Technical Report 172, Yale University, New Haven, CT.
- [5] Selfridge, M. (1981a). Why do Children Say "Goed"? A Computer Model of Child Generation. Proc. Third Annual Meeting of the Cognitive Science Society. Berkeley, CA.
- [6] Selfridge, M. (1981b). A Computer Model of Child Language Acquisition. Proc. 7th Int. Joint Conf. on Artificial Intelligence. Vancouver, Canada.
- [7] Selfridge, M. (1982). Why Do Children Misunderstand Reversible Passives? The CHILD Program Learns to Understand Passive Sentences. Submitted to the 3rd Annual AAAI Conference, Pittsburg, PA.
- [8] Wilks, Y. (1976). Parsing English II. In Y. Wilks and E. Charniak (eds) *Computational Semantics*, North-Holland Publishing Co., NY, NY

tially responsible for CHILD's robust performance. Finally, independent features make grammaticality judgements easy, since false features during understanding allow both a non-binary measure of the extent to which a slot filler was out of position, and also provide a basis from which to describe the incorrect position. Thus the first critical attribute of concept order syntax is the representation of syntax by independent features.

The second critical attribute of concept order syntax is that it is closely tied to word meaning. That is, concept order syntax is "semantically-indexed." This is reflected in two ways. First, syntactic features themselves specify where particular slot fillers within a specific word meaning are found. Second, the features sets are stored under the meaning of the word. Notice that these two aspects are not exactly the same thing. For example, one could have traditional grammar rules stored under word meaning, or one could have syntactic features stored not under word meaning but under the basic concept which comprises not only the meaning of a particular word but perhaps several others as well.

The third critical attribute of concept order is that it is derived from mechanisms which can plausibly be attributed to human language mechanisms. Specifically, it uses order information applied to concepts, slot fillers, and lexical items, a short-term memory, the mechanisms needed to retrieve the features and test them against the items in the short-term memory, and the concept of "slot" and "slot filler" applied to a knowledge representation. Now, humans can certainly perceive linear order, have concepts and lexical items, have the equivalent of a short-term memory, and the ability to test features against mental objects. It is somewhat less clear that they have memory containing conceptual case-frames. However, almost every proposal for modelling human memory uses a variant of this technique, and some have been very successful in accounting for psychological data (Anderson, Schank).

Describing natural language syntax using a representation which has the three attributes of a) independent features, a) semantic indexing, and c) psychologically plausible components thus seems to be largely responsible for the success it has had in accounting for the data that it does. Any extension to concept order syntax designed to enhance its ability to account for complex syntactic constructions should therefore maintain semantic indexing, independent features, and the primitive components which it currently uses. Probably such extensions must add additional primitive components; in this case they should be drawn from those plausibly attributable to the human cognitive system.

4.0 What Can't Concept Order Syntax Handle?

Experience with CHILD to date has shown the existence of a number of problems with its syntactic representation. This section will describe a subset of the known problems. First, concept order syntax cannot associate alternate meaning representations with alternative syntactic forms, or

feature sets. Thus a passive sentence is understood to mean exactly the same thing as its active version, while there is generally agreed to exist some minor meaning differences concerned with "focus". More drastic differences exist between declarative sentences and questions. Consider the difference between "That is Tom" and "Is that Tom?" If concept order syntax could associate different meaning with different syntactic feature sets, then it could possibly handle sentences like these.

The second problem is that of redundancy of storage of concept order syntax. Currently, all words which require syntactic information have that information stored directly associated with them, rather than in some central location. Thus currently concept order syntax manifests "syntactic lexical specificity." For example, object words such as "table" require syntactic knowledge to specify where in the input might appear fillers for their COLOR and SIZE slots. This information is stored directly under "table," associated with its meaning. Similar information is stored under every word whose meaning has COLOR and SIZE slots. This is done for several good reasons, most notably because it supports easy learning. Unfortunately, since every object word should have virtually the same syntactic knowledge stored under it, this scheme results in tremendous storage inefficiency. Worse, consider the process of learning a new word. One learns a new word after hearing it only once, yet immediately knows where in the sentence to find the fillers of the slots. It would appear that people's knowledge of syntax is stored in some more general way, and concept order syntax must be extended to handle this.

Third, concept order syntax cannot represent "syntactic embedding," in which a word has one kind of syntax if it is part of the top level of meaning representation, and somewhat different syntax if embedded within the meaning representation. Thus concept order syntax could not generate "I want to go to the store," but rather would say "I want that I go to the store." This particular problem is one which traditional theories of syntax are supposed to handle rather well.

Finally, concept order syntax has no ability to represent "agreement". For example, CHILD using concept order syntax would understand "the dog eats" and "the dogs eats" in exactly the same way, and would not be aware of any difficulty. Agreement is a relatively widespread linguistic phenomenon, and one which CHILD should certainly be able to learn.

5.0 Possible Extensions to Concept Order Syntax

Given the above shortcomings of concept order syntax, one would be tempted to abandon it as an extreme simplification and unworthy of further attention. However, since it has supported a high-performance model of child language learning, it appears worthwhile to explore directions in which it can be extended. The first extension is to couple syntax with meaning more tightly. This is apparently required by the problem that both active

In the initial experiment with a group of 15 year old pupils, LMS managed to produce a comparable diagnosis to investigators who interviewed the pupils several days later, [Sle 82b]. However, more recently we have used LMS with the same data base of rules with a group of 24 14 year old pupils and the results were dramatically different. LMS was unable to analyse a high proportion of the protocols as the appropriate mal-rules had not been predefined; see Figure 2.I for an example of a protocol which LMS could diagnose, and figures 2.II and 2.III for protocols which LMS could not diagnose. (Figure 2 reports several protocols collected during the pupil interviews; analogous protocols were collected with LMS, see [Sle ap] for details). The subsidiary difficulties which arise, if one does increase the number of mal-rules considerably is that the modeller then produces "multiple explanations for the same response, and secondly the speed of the on-line modeller drops dramatically as the search space becomes much greater. See [Sle 83] for a quantitative estimate for these effects. In the next section we discuss these issues in more detail.

2. An Analysis of the Difficulties Encountered

a) Multiple Explanations (Models) for a particular response One solution to this difficulty is to use an on-line task generator, to create a discriminatory task, [Sle 81b]. Alternatively, pupils could be asked to input the several intermediary steps in their solution in which case it might well be possible to discriminate between the several "illegal" paths. In particular the steps which were permissible need not be investigated only those where a correct transformation was not applied.

b) Speed of the on-line Modeller The first "solution" suggested above would not help the speed of the system, but the second might, as only those steps which had been done incorrectly need be considered.

c) On-line Modeller unable to analyse the protocol This is by far the most significant issue and the most challenging to resolve. We shall consider several approaches in varying degrees of detail.

1. Off-line analysis by the Investigator.
2. Computational aides to assist in the analysis by the Investigator.
3. On-line use of "schema" to generate mal-rules.
4. On-line inference of (mal) rules.

2.1 Off-line Analysis by the Investigator

The protocols given in Figure 2, and the additional protocols given in [Sle ap], were analysed in this manner. My initial reaction to several of these protocols was that they were totally random and only after subsequent analyses did I begin to see the consistent pattern which I have now recorded as part of each set. (Unfortunately it was not possible to talk with the pupils immediately following the sessions with

Page 6

LMS. When I did they explained to me very clearly what they were doing [Sle ap]). Because of the difficulties I encountered I implemented the Aide outlined in the next sub-section.

2.2 Aide to assist in the Analysis of Protocols

Having done a preliminary analysis of the protocols I supposed that many of them contain a

INFERRING (MAL) RULES FROM PUPIL'S PROTOCOLS

D. Sleeman

HEURISTIC PROGRAMMING PROJECT, [1]
 DEPARTMENT OF COMPUTER SCIENCE
 STANFORD UNIVERSITY
 CALIFORNIA, 94305.

ABSTRACT

This paper discusses protocols collected from 14 year-old pupils solving Algebraic tasks. In particular, it discusses several approaches to determining the mal-rule (ie an incorrect variant of a correct rule) which a pupil is using. The approaches considered include:
 -an off-line analysis by the investigator.
 -the definition of a number of perturbations, and attempts to see whether the pupil's answer can be explained by the application of one such perturbation.
 -and most significantly, the inference of mal-rules from protocols.

This latter approach is explored in some detail and it is shown how a rule-based system has successfully inferred the mal-rules encountered in earlier protocols. The importance of a rule inference system to a practical student modelling system is stressed.

1. INTRODUCTION

Several authors have pointed out the importance of having an accurate student model when one is attempting to give remedial teaching with an Intelligent Tutoring System. Indeed many of the papers in a recent collection address this issue. [Sle 82a]. One such system BUGGY/DEBUGGY [Bro 78], [Bur 82] attempts to model the student in terms of a predefined set of bugs and uses predefined heuristics to limit the size of the search space. Analogously, LMS - the Leeds Modelling System, uses a predefined set of rules and mal-rules (incorrect variants observed in

pupils' protocols) to define a space of possible student models. On the other hand a major emphasis with LMS has been the formulation of a search to "contain" the number of hypotheses/models considered. In this formulation, each set of tasks presented to the pupil focuses on a particular rule and it is shown that it is thus possible to dramatically reduce the search space. [Sle 81a] & [Sle 83].

LMS has been implemented as two phases. In the off-line phase, the system generates a complete and non-redundant set of models for each of the task types, and evaluates the predefined tasks against this list of models, [Sle 83]. (These models are sets of ordered productions rules, and the first rule which is satisfied fires. See figure 1a for a list of domain rules and 1b for a list of mal-rules used by pupils in protocols analysed earlier). In the on-line modelling phase the pupil is presented with a task and his response is evaluated against the various responses generated in the off-line phase. As it was anticipated that the users would only have minor manipulative errors, LMS allows the pupil merely to type in his final answer and none of the intermediary steps (he may give the intermediary steps if he wishes). LMS moves onto the next task when the pupil has given the solution in its simplest form, or has made an error or has exceeded a predefined limit for the number of refinement cycles allowed for each example type.

1. This work was done whilst I was at the University of Leeds, UK. Essentially, this paper previously appeared in the proceedings of the 1982 European Conference on AI, held at Orsay, France. It is intended to present recent developments of this work at the 1983 Machine Learning workshop.

In [Sle ap] I report 3 new mal-rules, variants of SOLVE, SIMPLIFY and BRA2, which can be explained by such a mechanism, as can most of the "original" mal-rules given in Figure 1b. Note that this schema would ALSO generate many mal-rules which I have NOT yet observed.

Although these schema help us understand the protocols given in Figure 2.II and 2.III, several of those reported in [Sle ap] can not be so explained, and so a further more general approach is needed.

2.4 On-line Inference of (Mal)-Rules

There are two principal disadvantages of the approach given in Section 2.2. Firstly, because it is not data-driven the search becomes very broad. Secondly, it relies on a pre-defined set of perturbations and hence would still be unable to spot very "wild" yet consistent, mal-rules. (An analogous criticism was also noted about the approach taken in section 2.3). For instance, during the 1981 experiment with LMS we noted the following mal-rules were used consistently:

$$\begin{aligned} MX+N=P &\rightarrow (M+N)X=P \\ M+NX=P &\rightarrow M+N+X=P \\ MX=N+P &\rightarrow MX=N \end{aligned}$$

Figure 4 shows a successful path for inferring the mal-rule for the example used in section 2.2. As can be seen from the figure, this process involves the following types of steps:

1. Using Rules in the Backward direction. (For example, the use of NTORHS in the Backward direction).
2. Use a series of heuristics to try to transform the sides of the equation into the same form as the "target" equation. This is the rationale of the

transformation applied between stages a to b, b to c, d to e and e to f.

3. When neither of the above can be applied then INFER a rule which completes the path (in this case between stages f and g). [In an earlier study, [Sle ap] we noted that the mis-parse rule appears to be applied to the task when it is first encountered, and so for the moment we shall assume that the inference step will be the last in the chain].

So far we have formulated 10 or so heuristics, a selection of which is given in Figure 5.

Note the following points about the algorithm:

1. There are an infinite number of integer pairs, (a b) which satisfy the relationship:

$$a + b = 1,$$
 where "i" is also an integer. However, by the use of the "Algebraic" form for "a" and "b", in HEURSUM1, and the subsequent instantiation of the integers to the RHS of the target equation, this set is reduced to a unique pair, which in the case of the example in Figure 4 is 6 and -8.
2. Figure 4 only shows a successful path. In general a search is involved as frequently several rules are satisfied by a particular state (eg both HEURSUM1 and HEURMULT1 are satisfied by a); and because a rule can be instantiated in several ways (eg the backwards form of NTORHS at c).
3. Some of the branching noted in the last point could be avoided by making the backward form of NTORHS more "sophisticated" by allowing it to access the target equation.

"wild" step, and so I postulated a number of "perturbations" including:

$$MX \rightarrow M + X \dots \dots \dots [P1]$$

$$MX \rightarrow MX + M \dots \dots \dots [P2]$$

A system has been implemented which attempts to transform the original task into the pupil's answer using only one "perturbation", but as many applications of the domain's rules and mal-rules as necessary, per "path". Figure 3 gives two examples of such paths. [NB, we are NOT claiming that students undertake all the steps given in Figure 3, but merely that their answers are as if they did. Deciding the particular "grain" size DOES become an important issue when one uses the inferred model as a basis for student remediation.]

Given a reasonable number of such perturbations it is clear that the search space becomes extremely large. However, it should be added that this sub-system did provide several explanations which I had not previously considered.

2.3 Schema to generate Mal-Rules (On-line)

When carrying out interviews with the 1981 group of children some of them gave explanations for their actions which amounted to schema for creating mal-rules.

In figure 2.III we gave a substantial section of pupil AB18's protocol. In the course of our discussion he explained that he was carrying out the teacher-given algorithm of:

"Collecting all the Xs on the left hand side and collecting all the numbers on the right hand side." Pupil AB17, figure 2.II, gave a similar explanation for his action, and added that he was not really sure what to do about the "extra multiply signs".

This gives us a schema for generating mal-rules. For example given the task type:

$$M \times X + N \times X = P$$

This schema gives the following "action sides" for mal-rules:

$$X + X = P - M - N$$

$$X + X = P + M + N$$

where in the second case the X coefficients are treated "specially", ie the coefficients of the Xs were taken across to the RHS of the equation but the signs were NOT changed. And the form given by pupil AB-17, and quoted in Figure 2.II, namely:

$$*X + X = P - M - N$$

which he went on to "interpret" as:

$$X + X = P - M - N$$

Its "complementary" form being:

$$X + X = P + M + N$$

Similarly, given the task type:

$$M \times X = N \times X + P$$

This schema creates the following forms:

$$X = N + P - M$$

$$X = N + P + M$$

$$X + X = N + P - M$$

$$X + X = N + P + M$$

$$X - X = N + P - M$$

$$X - X = N + P + M$$

For example on task h, pupil AB18 suggested the use of both the third and the fifth forms (see figure 2.III).

Analogously, it appears possible to generate all the mal-rules associated with a manipulative rule by systematically removing one or more of the rule's sub-steps, [2].

2. Brown and Burton's modelling work [Bro 78] assumed that pupil errors were due to perturbations of the correct rules (cf the assumptions which we also made initially in this project, and which seems to be true for some of this domain's rules). This however, is certainly not true for the class of errors which we are discussing here. Indeed, pupils AB-18 and AB-17 do not use the usual "task segmentation" and thus do not even consider applying, at the first stage, a correct rule.

Where M, N and P are integers and where lhs, rhs etc are general patterns (which may be null), where +|- means either + or - may occur, where SHD indicates the String Head and where < and > represent standard "algebraic brackets". (NB the respective Rule CONDITIONS and ACTIONS are on alternate lines).

b) Some MAL-RULES for the Domain

RULE NAME	CONDITION/ACTION
MSOLVE	(SHD H*X = N) (SHD X=M/N) or (SHD INFINITY)
MNTORHS	(lhs + - M = rhs) (lhs = rhs + - M)
M2NTORHS	(lhs1 + - M lhs2 = rhs) (lhs1 + - lhs2 = rhs + - M)
M3NTORHS	(lhs1 + - M lhs2 = rhs) (lhs1 + - lhs2 = rhs + - M)
MXTOLHS	(lhs = + - M*X rhs) (lhs + - M*X = rhs)
M1BRA2	(lhs M *<N*X + - P> rhs) (lhs M*N*X + - P rhs)
M2BRA2	(lhs M*<N*X + - P> rhs) (lhs M*N*X + - M + - P rhs)

Using the same conventions as above.

FIGURE 2

Three examples of very consistently used MAL-RULES.

I) Pupil AB-11 on task set 7.

a) The task given was: $4 + 2*X = 16$

Pupil writes
1) $6X = 16$
2) $X = 2.6666$

b) The task given was: $2 + 4*X = 14$

Pupil writes
1) $6*X = 14$
2) $X = 2.333$

c) The task given was: $3 + 5*X = 11$

Pupil writes 1) $8*X = 11$
(and is told to leave it in that form)

d) The task given was: $5 - 3*X = 11$

Pupil writes 1) $2*X = 11$
(and is told to leave it in that form)

II) Pupil AB-17 on task set 6

a) The task given was: $2*X + 4*X = 12$

Pupil writes
1) $* X * X = 12 - 2 - 4$
2) $X ** 2 = 8$
3) $X = \text{ROOT } 8$

b) The task given was: $2*X + 3*X = 10$

Pupil writes
1) $* X * X = 10 - 2 - 3$
2) $X ** 2 = 5$
(and is told to leave it in that form)

c) The task given was: $2*X - 3*X = 10$

Pupil writes
1) $* X * X = 10 - 2 - 3$
2) $X ** 2 = 11$
(and is told to leave it in that form)

III) Pupil AB-18 on task sets 5, 6, 7 and 8.

a) The task given was: $2*X + 3*X = 10$

Pupil writes
1) $2*X = 10 - 2 - 3$
2) $2*X = 5$
3) $X = 2.5$

Page 19

b) The task given was: $3*X + 5*X = 24$

Pupil writes
1) $X + X = 24 - 3 - 5$
2) $2*X = 16$
3) $X = 8$

c) The task given was: $3*X + 4*X + 5*X = 24$

Pupil writes
1) $X+X+X = 24-3-4-5$
2) $3*X = 12$
3) $X = 4$

d) The task given was: $2*X + 4 = 20$

Pupil writes
1) $X = 20 - 2 - 4$
2) $X = 14$

e) The task given was: $3*X + 5 = 7$

Pupil writes
1) $X = 7 - 3 - 5$
2) $X = -1$

f) The task given was: $4 + 3*X = 14$

Pupil writes
1) $X = 14 - 3 - 4$
2) $X = 7$

g) The task given was: $5 + 6*X = 20$

Pupil writes
1) $X = 20 - 5 - 6$
2) $X = 9$

h) The task given was: $4*X = 2*X + 6$

Pupil writes
1) $2*X = -4 + 2 + 6$
2) $2*X = 4$
3) $X = 2$

Pupil then wrote
1) $X - X = 2 + 6 - 4$
2) $0 = 4$
and QUITs.

i) The task given was: $5*X = 3*X + 6$

Pupil writes
1) $0 = 4$
and QUITs.

FIGURE 3.

$3X + 5 = 6$	$3X + 5 = 6$
$\Rightarrow [P1]$	$\Rightarrow [P2]$
$X + 3 + 5 = 6$	$3X + 3 + 5 = 6$
$\Rightarrow [ADDSUB]$	$\Rightarrow [ADDSUB]$
$X + 8 = 6$	$3X + 8 = 6$
$\Rightarrow [NTORHS]$	$\Rightarrow [NTORHS]$
$X = 6 - 8$	$3X = 6 - 8$
$\Rightarrow [ADDSUB]$	$\Rightarrow [ADDSUB]$
$X = -2$	$3X = -2$

This figure gives the TRACES for 2 paths one involving the perturbation P1 ($MX \Rightarrow M + X$) and the other P2, ($MX \Rightarrow MX + M$). The implemented algorithm returns a list of all the TRACES which explained the pupil's response, say $X = -2$. Where $\Rightarrow$ implies that the "upper" form is transformed into the "lower".

4. A further improvement in the efficiency can be achieved by segmenting the rule-set into related sub-sets, and only having the relevant sub-set in scope at any time.

5. If we look more closely at the inference step we see that it is possible to infer that

$$MX + N = P \Rightarrow X + M + N = P$$

where M, N and P are all integers. However, if the algorithm notes which symbols have remained unchanged then it is possible to infer that the substantive change is:

$$MX \Rightarrow M + X$$

Neves, [Nev 78], uses a similar approach to deduce the essential features of rules given the several steps of a solution culled from an Algebra text.

6. The instantiation heuristics are a little more complex than the rules given in Figure 5.

3. Current Activities

Many of the protocols collected in the recent experiment, [Sle ap], have been analysed, using the approach outline in the last section. To test the generality of this approach we have also used it to infer the NTORNS rule, see Figure 6.

ACKNOWLEDGEMENTS

The pupils of Abbey Grange School, Leeds provided some fascinating protocols. Jaime Carbonell, Pat Langley, K. Lovell, Steve Mallard and Tim Hepworth for discussions concerning this work. Alan Bundy suggested some significant clarifications to the ECAI version of this paper.

REFERENCES

- [Bro 78] J.S. Brown & R.R. Burton (1978). Diagnostic Models for procedural bugs in basic Mathematical Skills, in *Cognitive Science*, 2.2, pp 165-192.
- [Bur 82] R.R. Burton. (1982). Diagnosing bugs in simple procedural skills. In *Intelligent Tutoring Systems*, edited by D.H. Sleeman and J.S. Brown, published by Academic Press, pp167-183.
- [Nev 78] D.M. Neves. (1978). A computer program that learns algebraic procedures by examining examples and working problems in a textbook. *Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence*, pp191-195.
- [Sle 81a] D.H. Sleeman & M.J. Smith, (1981). Modelling student's Problem Solving. *Artificial Intelligence*, pp 171-187.
- [Sle 81b] D.H. Sleeman. (1981) A Rule-based Task Generation System. In *IJCAI Proceedings*, pp 882-887.
- [Sle 82a] D. Sleeman & J.S. Brown, (1982) Editorial in *Intelligent Tutoring Systems*, edited by D.H. Sleeman and J.S. Brown, Academic Press, pp 1-12.
- [Sle 82b] D. Sleeman. (1982) Assessing competence in Basic Algebra. In *Intelligent Tutoring Systems*, edited by D. Sleeman and J.S. Brown, Academic Press, pp 186-199.
- [Sle 83] D.H. Sleeman, (1983). A rule directed Modelling system. In *Machine Learning*, R. Michalski, J. Carbonell, and T. M. Mitchell, (editors), Tioga Press, pp483-510.
- [Sle ap] D.H. Sleeman. (in press) Basic Algebra revisited: a study with 14 year olds. *Journal of Mathematical Behaviour*. (Also published as Stanford HPP memo 83-9).

FIGURE 1
a) RULES for the ALGEBRA domain
(evaluative form and slightly stylized).

RULE NAME	CONDITION/ACTION
FIN2	(SHD X = M/N) (SHD (M N)) or (SHD evaluated)
SIMPLIFY	(SHD X = M/N) (SHD X = M'/N')
SOLVE	(SHD M * X = N) (SHD X=N/M) or (SHD INFINITY)
ADDSUB	(lhs M + - N rhs) (lhs [evaluated] rhs)
MULT	(lhs M * N rhs) (lhs [evaluated] rhs)
XADDSUB	(lhs M*X + - N*X rhs) (lhs (M + - N) * X rhs)
NTORNS	(lhs + - M = rhs) (lhs = rhs - + M)
REARRANGE	(lhs + - M + - N*X rhs) (lhs + - N*X + - M rhs)
XTOLNS	(lhs - + - M*X rhs) (lhs - + M*X = rhs)
BRA1	(lhs < N > rhs) (lhs N rhs)
BRA2	(lhs M*<N*X + - P> rhs) (lhs M*N*X + - M*P rhs)

Domain-specific Learning and the Subset Principle

Robert C. Berwick

MIT Artificial Intelligence Laboratory, Room 820
545 Technology Sq., Cambridge, MA 02139

Abstract

A key issue for learning theory has been the relative importance of domain independent learning vs. domain specific learning. How much of learning is attributable to general learning methods like "inductive generalization" and how much is attributable to particular techniques and representations that apply only to specific domains? This paper explores this issue by analyzing the role of one general principle in the context of several very specific domains. Angluin [1980] established a necessary and sufficient condition for the acquisition of a (recursive) language (in the sense of Gold [1967]) from positive-only evidence. The effect of this condition is to impose an ordering on possible sequences of guesses about the target language combined with the array of possible data sequences in such a way that the acquisition system always guesses the narrowest possible language compatible with the data given so far. In this paper, this "subset principle" is applied to several areas where extensive domain-specific knowledge is present, including syntactic constructions and phonological distinctive feature systems. Several well-known constraints on these systems are shown to follow the subset principle, providing some evidence that natural learning systems are designed so as to be easily learnable.

Key Terms: Mathematical learning theory, concept acquisition, language acquisition.

account for the shape, structure, and function of the kidney, the liver, the heart, the visual system, and so forth.

Chomsky [1980] p. 245.

The challenge for the learning theorist is to come up with some general principle that actually counters Chomsky's view. It is harder to do this than one might think; the remainder of this paper is an attempt to discuss one leading contender for a true general principle of learning -- the Subset Principle. In the next section this principle is defined and illustrated formally. As we will see, the effect of the Subset Principle is to impose an ordering on the way that (positive only) evidence is presented to a learner. This restriction shows up in a surprising number of domain independent and dependent contexts; these are taken up in Section III. Among the implications of the principle are: the difficulty of acquisition of disjunctive concepts; Keil's [1979] "M constraint" on predicability trees (roughly, natural semantic networks); the design of human phonological systems and their acquisition; and the existence of otherwise mysterious constraints on natural language syntactic systems. The good news, then, is that there seems to be at least one principle of learning that applies "across the board"; the bad news is that it is a necessary and sufficient condition for acquisition from positive-only evidence -- so it is no surprise that it is obeyed.

II. THE SUBSET PRINCIPLE

We will approach the formulation of the Subset Principle in a somewhat round-about way. Suppose we try to see what blocks simple, incremental, learning procedures from working. The thorn in the side of many learning procedures is over-generalization; the Subset Principle removes that thorn. More particularly, a major and long recognized problem with the use of disjunctions in what is learned is that the danger of over-generalization becomes more acute. By definition, the conjuncts of a disjunctive description have no features in common (save for some very general root predicate). This fact is the bane of simple generalization procedures. For if we attempt to establish one half of the disjunctive clause by observing a positive example, and the other half by another example, then a simple-minded generalizer will often conclude that the correct description of the examples is the intersection of these individual descriptions, namely, the null set. For example, if the correct description of an object is $P \vee Q$, where P and Q are themselves conjunctive expressions, then one positive example can be P , and another Q ; the intersection of the two can be empty.

There are two basic solutions to the problem of disjunction. One is to admit negative evidence, so that one can know if a description is too general. That is the function of the G boundary set in the version space model of acquisition: it delimits the outer boundary of possible descriptions compatible with examples seen so

I. INTRODUCTION

What is the scope and power of "general" principles of learning? Traditionally, general methods like "inductive generalization" or "analogy" have figured prominently in any discussion of learning. And those who have embraced such general learning principles have ranged across the scientific spectrum, from Skinner to Piaget to Newell and Simon. More recently though, the mere existence of general learning principles has come under serious question:

What I intend to signify in referring to the doctrine of uniformity of mind is...that there are general principles of learning that underlie all of these systems, accounting for their development; "multipurpose learning strategies," as they are called, that apply "across the board." In contrast, it might be proposed various "mental organs" develop in specific ways...and that multipurpose learning strategies are no more likely to exist than general principles of "growth of organs" that

FIGURE 4

g) $3X + 5 = 6$
 $\rightarrow$ [Inference step]
 f) $X + 3 + 5 = 6$
 $\leftarrow$ [Instantiate to LHS of target eqn]
 e) $X + [6-n] + [n] = 6$
 $\leftarrow$ [HEURSUM2]
 d) $X + 8 = 6$
 $\leftarrow$ B[NTORHS]
 c) $X = 6 - 8$
 $\leftarrow$ [Instantiate to RHS of Target equation].
 b) $X = [-2-n] + [n]$
 $\leftarrow$ [HEURSUM1]
 a) $X = -2$

Shows a scheme for inferring the Mal-rule $MX \rightarrow M + X$ given the task $3X + 5 = 6$ and the pupil's response $X = -2$. Where B[R] implies use of the rule, R, in the Backwards direction, where $\rightarrow$ is as in Figure 3 and $\leftarrow$ implies that the "lower" form is transformed to the "upper" one.

FIGURE 5.

```

HEURSUM1
IF lhs(eqn) = lhs(target)
AND rhs(eqn) = rhs(target)
AND rhs = 1
THEN replace(1, [1-n] + [n])

HEURSUM2
IF lhs(eqn) = lhs(target)
AND rhs(eqn) = rhs(target)
AND contains(lhs, 1)
THEN replace(1, [1-n] + [n])

HEURMULT1
IF lhs(eqn) = lhs(target)
AND rhs(eqn) = rhs(target)
AND rhs = 1
THEN replace(1, [1*n] / [n])

```

Shows a selection of the heuristics which have been implemented: the first 2 were used in the protocol given in Figure 4. All the heuristics, including the more complex instantiation rules have been implemented as Production Rules. (Here "target" is short for TARGET equation, eg $3X + 5 = 6$ in Figure 4, and "eqn" is short for the current equation, which in Figure 4 is initially, $X = -2$).

FIGURE 6.

d) $2X + 3 = 9$
 $\rightarrow$ [infer rule]
 c) $2X = 9 - 3$
 $\leftarrow$ [Instantiate to RHS of Target].
 b) $2X = [6-n] + [n]$
 $\leftarrow$ [HEURSUM1]
 a) $2X = 6$

Showing the inference of
 $MX + M = P \rightarrow MX = P - M$
 namely the NTORHS rule, given that the original rule-set contained only the SOLVE FIN2 and ADDSUB rules.

Given a family of recursive languages, L . L is identifiable from positive-only evidence iff there exists an effective procedure which on input i enumerates finite sets $T_1, T_2, \dots$ such that

- (i) $T_i \subseteq L_i$
- (ii) For all $j > i$, if $T_j \subseteq L_j$, then L_j is not a proper subset of L_i .

The intent is for the T_j to be "trigger" sets for hypotheses, and the condition amounts to the claim that such finite triggers must exist for positive identification to work.

Since the principle is a necessary condition for positive-only acquisition, it is not surprising to discover that it subsumes a variety of proposals that have been advanced in the linguistic literature as constraints on the order in which acquisition hypotheses should be made. In fact, it appears as though the Subset Principle exhausts what can be said about ordering constraints in acquisition. To support this claim we will review several proposals that have been made regarding the ordering of hypotheses in acquisition, some in detail.

III. APPLICATIONS OF THE SUBSET PRINCIPLE

3.1 The acquisition of phonological segments

According to the distinctive feature theory of phonology originally developed by Prague school structuralists such as Jakobson and pursued by Chomsky and Halle in *Sound Pattern of English* [1968], all natural sound segments such as *a* or *p* can be described via a small number of binary-valued distinctive features. By and large these features have an articulatory or an acoustic grounding, with names suggestive of their place or manner of articulation, such as *high*, *back*, *anterior*, and the like, 20 or so features in all. Given binary values for distinctive features (+ or -), there are 2^{24} possible segments (about 16.8 million). However, most of these segments are not attested in human phonological systems. Furthermore, most collections of segments in a particular natural language do not make use of an extensive number of these possible contrasts.

As Kean [1974] shows, in part the reason for this is that distinctive features are not fixed in isolation. Rather, certain distinctive features can be fixed *only after* certain other features are fixed. For instance, according to Kean's theory the distinctive feature *consonantal* must be fixed before the feature *back* or *continuant*.

Kean developed this theory as way to explain some of the observed restrictions on possible segmental systems and possible phonological rule systems. But there is another way to interpret such a theory, and that is as a developmental program for how a segmental system is acquired. That is, we can identify the *markedness* of a particular rule, segment, or phonological system as the amount of information required to fix that system developmentally. This structure in turn permits a considerable compression of the information required to fix a particular segmental system. Importantly, the system obeys the Subset Principle. By construing the theory in this new way, one can in fact exhibit an acquisition system in which large numbers of developmental pathways are eliminated because of the *order* in which a small number of parameters are set. Moreover, the developmental model actually *explains* the existence of Kean's constraints, constraints that, while presumably universal, are

otherwise not accounted for. Put another way, taken as a model of acquisition the Kean theory provides an explicit example of how developmental sequencing can actually be an aid for acquisition, above and beyond the constraints provided by models of acquisition that assume an "instantaneous" fixing of a system of knowledge.

According to distinctive feature theory, segments can be distinguished only if they have different values for at least one of the 24 distinctive features. For example, the segments *a* and *i* are distinguishable given a segmental system that sets all distinctive features for the two segments to the same value save for the feature *back*. *a* is unmarked for back, while *i* is marked for back (i.e., is expected). To take another example, the segment *ae* is also marked for back (-back) but is distinguishable from *i* because it is additionally marked +low.

What Kean established is that there is a certain *logical* dependency order in which segmental distinctive features are fixed. For example, a segment must be first marked as consonantal or not; then, once the value for the feature *cons* has been set, value for the distinctive feature *anterior* may be fixed. Now, Kean deduced this order by examining the array of possible phonological systems in natural languages. More interestingly though, if we regard this order as a model of how a sound system is actually acquired, then we get a picture like this:

Hierarchy diagram of example class splits.

Step 1	+syl		+syl
	+cons=unmarked cons	m cons	+cons
	cons		m
	[p, n, m, k]	[h, ?, w]	[t, k, k ^w , r,] [h]
Step 2	mark anterior		mark anterior
	(trigger: k)		(trigger: k, k ^w)
	u anterior	m anterior	u anterior
	anterior		m
	[p, n, m]	[k]	[t, r,] [k, k ^w]
Step 3	mark sonorant		mark sonorant
	(trigger: n, m)		(trigger: r)
	u sonorant	m sonorant	u sonorant
	sonorant		m
	[p]	[m, n]	[t] [r]
Step 4	mark labial		mark labial
	(trigger: m)		(trigger: k ^w)
	u labial	m labial	u labial
			m labial
	[n]	[m] (but also	[k] [k ^w]
		[p])	

Different segmental systems correspond to different patterns of developmental paths. Note that at any given step where a class split occurs one must determine whether the class that is *not* prompting the split is marked or unmarked at step $i+1$. Thus [p] is marked labial at step 4 even though it doesn't prompt the split into marked/unmarked labial.

far by describing the properties of negative examples. But what if negative examples are not available? The only other possibility is to eliminate disjunctive statements.¹

In the version space framework, the non-identifiability of disjunctive languages is revealed by the structure of the lattice representing possible identification hypotheses. Since we are using only positive evidence, we cannot move the boundary set G downwards; only the lower bound can be moved upwards. No matter what possible hypothesis is made at any step i , the acquisition procedure can be forced to make a more refined guess at some later step, moving it to a broader class of languages, simply by adding an example that was not in the language guessed so far. But since the upper bound G cannot be moved, we can only approach G by moving the boundary set S upwards, one step at a time. Each move represents a new guess; since there are a countably infinite number of moves between S and G initially, the acquisition procedure must make an infinite number of guesses, violating identification in the limit. The crux of the problem is that the space of hypotheses is "dense" in the sense that there are collections of positive evidence compatible with an infinite number of possible guesses. Note that this is the reason that the example family of languages provided by Gold [1967], the family of all languages of finite cardinality and one language of infinite cardinality, fails to be identifiable from positive-only evidence: there is a finite collection of evidence sentences that is covered by an infinite number of languages in the family.

1. How much disjunction is too much? The following example shows that if "arbitrary" disjunction is allowed then there exists an infinite collection of languages whose individual members cannot be identified from positive-only evidence.

Example. (Angluin, [1980])

Suppose that a family of languages $\mathcal{L}$ includes the "universal" language Σ^* . In addition, suppose that $\mathcal{L}$ contains languages that are defined by rule systems that can express arbitrary disjunctions. That is, suppose that grammars for these languages can be of the form $A \rightarrow \alpha[\beta_1 \dots \beta_n]$, with no restrictions on the right-hand sides α, β (the rewriting system is still finite in size, of course). The intent is that each part of the right-hand side of the grammar covers one sentence in L_i . In other words, we assume the power to form the language of any member of this family as the union of individual sentences of that language. Let us call these languages the *disjunctive covering languages* of $\mathcal{L}$.

From the standpoint of acquisition, the problem with this family of languages is that any finite collection of samples from a language in $\mathcal{L}$ is covered by two languages: the universal language and the language generated by the grammar, $S \rightarrow L_1 \vee L_2 \vee \dots \vee L_n$. But then there is a sequence of positive examples drawn from a member of $\mathcal{L}$ that will cause an acquisition procedure to change its guess about the identification of L_i an infinite number of times, and thus the procedure will fail to identify the members of $\mathcal{L}$ in the limit. The sequence is simply members drawn from the universal language. At each step i , an acquisition procedure must guess that examples it has seen have been drawn from either the universal language or the disjunctive covering languages of the examples it has seen. Suppose the procedure guesses the universal language at point i ; then no examples it sees later will force it to change its guess, since no example will contradict the guess it has just made. There is some disjunctive cover language whose examples are just like those of the sequence seen up to point i : simply conjoin the examples seen so far, disjunctively; call this language DL_i . But then the procedure must fail to identify DL_i , since there is the sequence of data consisting of the sequence that forced the procedure to guess the universal language followed by examples drawn just from DL_i on which the procedure does not identify DL_i in the limit. Here we make use of disjunctive power. Conversely, if the acquisition procedure guesses DL_i at point i , one can simply add an example that is in $UL - DL_i$; here we make use of the density of UL . Since we have assumed a countably infinite number of DL_i , the procedure can be made to change its guess an infinite number of times. (This example is a generalization of Gold's [1967] result that any family containing all the finite languages and one infinite language is not identifiable in the limit from positive-only evidence.)

In short, there are two general situations in which an incremental, conservative approach to acquisition will work:

(1) If positive and negative examples are available, then a partial ordering of hypotheses such that the accumulation points of the ordering are themselves hypotheses suffices. (This is the version space approach.)

(2) If just positive examples are available, then a sparse covering of evidence sets suffices. That is, only a finite number of hypothesizable languages L_i can cover any possible evidence set. If general disjunctive statements are allowed -- then this condition cannot, in general, be met. (It might be that some other condition could suffice, however.)

The crucial condition required to ensure identifiability from positive evidence may be stated in another way. The problem with overly-dense families of languages is that, no matter what the positive evidence set, so that data set D_i prompts a guess of language L_i , it is always possible to find another language L_j that is compatible with D_i and can be interposed between D_i and L_i . But suppose that this never happens, that is, that an L_j can never be so interposed. Then either one of two situations must obtain, $L_j \supset L_i$, or L_i and L_j are incomparable, but both cover the data set D_i . In case 1, either L_j is identical to L_i -- hence cannot prompt a hypothesis change, under the Gold assumptions -- or $L_j - L_i$ is non-empty, and there is some d in L_j not in L_i . If L_j is the target language, then this example must eventually appear in the information presentation (again under the Gold assumption of a complete presentation), and an identification procedure will have evidence to change its guess. If L_i is the target language, then the procedure will never be forced to change its guess. In case 2, the same situation obtains. If either L_j or L_i is the correct target language, then some positive example must appear that indicates this possibility -- a sentence not in L_i , but in L_j , or a sentence not in L_j , but in L_i . (This is where the recursiveness of the L_i is used.) Therefore, in either case identification in the limit from positive evidence can proceed successfully. Let us say that when a family of languages meets this condition that it obeys the Subset Principle. The point of this condition is to ensure that the acquisition procedure always guesses a subset language if possible, that is, the smallest language that is also compatible with the positive evidence so far encountered. (Note that the sequence of guesses conforms to an incremental search from more specific to less specific languages if the lattice of languages is ordered by the subset relation. Again, what is crucial to the demonstration is the topological structure of the lattice of hypotheses. The particular details of what these hypotheses denote is irrelevant.)

The Subset Principle was independently observed and formalized in a recursive-function theoretic framework by Angluin [1980]. In an important theorem, Angluin demonstrated that the Subset Principle is necessary and sufficient for identifiability from positive evidence.

This developmental assumption is just like that of the phonological model. A new phonological partition is formed when it is realized that a segment is *not* like the segments of its classmates, and hence must be marked. (The hierarchy tree dictates *how* it must be marked.)

There are other similarities between this situation and the case of phonological systems. With rare exceptions, children were found to obey the M-constraint. This constraint is also honored by phonological rules. Further, the lattice of predication had an upper bound, an all-embracing predicate that could be applied to any object -- e.g., "think about". The corresponding predicate in the phonological domain is "syllabic" or, perhaps, "segmental".

Again, it is this Subset Principle that is being obeyed here. If refinement occurs at the "leading edge" of the hierarchy tree, then the new representation is a minimal refinement of the old one, in the narrowest possible way. This kind of conservative strategy is not necessary -- unless one is faced with the problem of acquisition based on just positive evidence. Thus the widespread existence of this strategy might be taken as an indication of a reliance on positive evidence here (as in the phonological case).

3.3 Some examples from syntax

The Subset Principle is also ubiquitous in the acquisition of the syntax of natural languages. A catalogue includes (but is not exhausted by) the following examples. In fact, as far as can be determined, the Subset Principle subsumes all the specific ordering principles advanced in current syntactic theory.

3.3.1 Bounding nodes for subadjacency

In most current theories of generative grammar, one assumes that movement rules obey a certain "locality principle", in that a movement cannot cross more than a single clause boundary -- the Subadjacency constraint.

John is certain [_S trace to like ice cream]
 *John seems [_S it is certain [_S trace to like ice cream]]

The man [_{wh}_i [_S I don't know [_{wh}_j [_S
 trace_j knows trace_i]]]]

Interestingly, this last sentence is grammatical in Italian, as discussed by Rizzi [1978]:

L'uomo [_{wh}-phrase_i che [_{non} so [_{chi}_j [_{trace}_j conosca trace_i]]]]

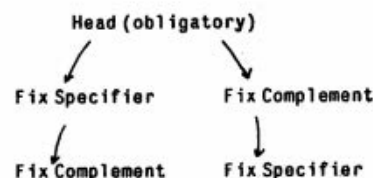
According to Rizzi, this is because it is *S*, not *S*, that is a bounding node for Subadjacency in Italian. Therefore, the empty category *trace_j* can be co-indexed with the *wh*-phrase_i because it crosses only a single full clause boundary; the second boundary is an *S*, not an *S*. Apparently, the choice of a bounding node is yet another parameter that must be set in order to "fix" a grammar.

Suppose Rizzi's analysis is correct. How could the choice of bounding node be determined on the basis of evidence received by an acquisition procedure? Once again, let us apply the Subset Principle. If the bounding node for Subadjacency is *S*, then a narrower class of languages is generated than if the bounding node for Subadjacency is *S*. Therefore, by the Subset Principle, the acquisition procedure's first hypothesis should be to set the bounding node for Subadjacency to *S*. In other words, the default assumption is that all languages are like English in this regard. If

this assumption is wrong, then a positive example will appear that violates *S*-bounding -- as in the Italian example above. Then the acquisition procedure can reset the Subadjacency parameter to the next "largest" value, namely, *S*.

3.3.2 Acquisition of phrase structure rules

Williams [1981] describes a special case of the Subset Principle as a potential model for the acquisition of the context-free phrase structure rules that underlie the basic sentence forms of a language. The idea is simple. According to current linguistic theory, a phrase structure rule obligatorily contains a *Head*, a unit of the same type as the phrase itself, and optionally has *Complements* and *Specifiers*. For example, in an English Verb Phrase the Verb itself is the ultimate (lexical) *Head*; the trailing NPs and PPs, and *S* are the *Complements*; perhaps *Adverbs* are *Specifiers*. Given this model, then the Subset Principle dictates a particular order in which hypotheses should be entertained. The first assumption is to assume that a phrase has *no* *Specifiers* or *Complements*, since this generates the narrowest possible class of languages. As usual, this initial assumption would be dropped in the face of positive evidence to the contrary -- in this case, the existence of a *Complement* or a *Specifier*. Thus the order of hypotheses would be as follows:



As Williams observes, there is no particular order in which specifiers or complements are fixed, relative to each other. Further, since either or both can be missing, the end state may be a phrase structure rule lacking in either *Specifiers* or *Complements*. More often, however, *Complements* are obligatory arguments; hence the general order of acquisition will be to fix *Complement* structure before fixing optional *Specifiers* (as also determined by the Subset Principle; see the discussion of obligatory arguments immediately below).

3.3.3. Optional and obligatory arguments and rules

More generally, it has been widely pointed out (see e.g., Baker [1979]) that it is difficult, if not impossible, to determine whether a rule is obligatory or optional on the basis of just positive evidence. For instance, Baker observes that Subject-Auxiliary inversion is obligatory in main clauses. If an acquisition procedure guessed a Subset language and assumed that inversion was optional, it could generate two forms, the second ungrammatical:

Who will John kiss?
 Who John will kiss?

Suppose counterfactually that the inversion rule was optional in main clauses. This causes no problem; the acquisition procedure will simply acquire a new rule of analysis to handle that situation as well. In short, one can see that by formulating the acquisition

The important feature of the partitioning process is that splitting occurs at the leading edge of the directed hierarchy graph, by successive refinement of existing classes of segments. Because extension of classes occurs solely via the refinement of existing partitions, the set of segmental classes at step i will be a homomorphic image of all of the other trees before it in the developmental sequence. Also note that the tree is developed by marking just one distinctive feature at a time -- not a necessary constraint, since it is not clear why one could not develop a new class by marking two or more features in one step. The result of this constraint is to guarantee that at any step i in the development of a segmental system the classes of segments will be at most one mark (m) different. For instance, this constraint excludes the following array of marks:

	X1	X2	X3	X4
Feature:				
consonantal	m	u	u	u
anterior	m	u	u	u
back	m	u	u	u
low	u	m	u	u
labial	u	u	m	u
sonorant	u	u	m	u

From one point of view the one-mark constraint is a puzzling one. It is not at all obvious why segmental systems should be designed so that the alteration of a single distinctive feature could convert an a into an a^2 .

The key observation is that the one-mark constraint follows the Subset Principle -- and so then do natural segmental systems. If this observation is correct, then the one-mark constraint fits naturally into a model of incremental acquisition from positive-only evidence (rather than a model of error detection or correction). Note also that marking is always carried out at the immediate fringes of the existing class tree.

To see how important this is, suppose that acquisition is not incremental, so that two or more marks can be added at a single step. It would then be possible to form a new class partition based on marking both the features *labial* and *sonorant* without having first used the feature *sonorant* to form any segment classes. Now there is no class that will correctly accommodate a segment that is [u labial, m sonorant]. One way to remedy this problem would be to allow the procedure to go back and rebuild classes that have already been formed, but this would violate the developmental ordering that has been assumed. The fringe of the hierarchy tree would no longer summarize the possible next states that could be hypothesized, since there could be segments such as n that would demand the interpolation of new classes between older partitions and the current partition. In other words, the one-m constraint amounts to the demand that new classes be the minimally specific refinements of existing classes. This way of ordering hypotheses in a lattice of general to specific descriptions is, as we shall see immediately below, common to several proposed models of so-called "concept acquisition." It is also sufficient to guarantee that the incremental acquisition procedure proposed earlier will work correctly.

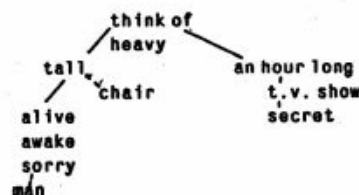
2. Note that this would seem to be an unwise design choice from the standpoint of error detection or error correction; as is well known, in order to be able to correct errors of k bits, then segments would have to be separated by a ball of radius $2k+1$ (since one must guarantee that changes of up to k bits in any two segments still leave one able to determine the original segment).

Put another way, the class of possible natural segmental systems seems to be more restricted than one might expect given a model where segments are obtained by selecting arrays of marked and unmarked distinctive features at random. Segmental systems are constrained in just the ways one might expect if they were acquired on the basis of an incremental fixing of segments according to the Kean feature hierarchy.

3.2 Concept acquisition and the M constraint

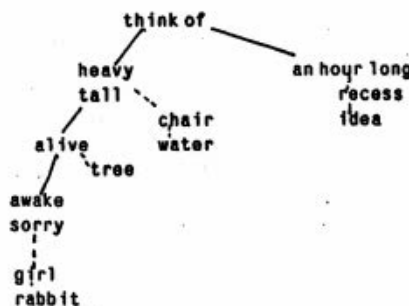
Related research on conceptual development as pursued by Sommers [1971], Keil [1979], and others also implicates the Subset Principle. Briefly, Sommers has suggested that if one arranges terms such as *dog*, *cat*, *love*, etc. into a graph structure whose terminal leaves are terms and where each node is an item that can sensibly be predicated of the all terms below it in the graph, then one never, or rarely, finds human intuitions of sensibility resulting in "M" shaped graphs -- the so-called "M" constraint. Rather, the graph structures take the form of hierarchical trees. For instance, the tree below (reproduced from Keil [1979, page 15, Figure 1]) is typical of one that comports with human intuitions; an "idea" can be predicated as being true, hence lies under a node labeled "true" in a predication tree, but cannot be at the corner, or red, hence does not lie under nodes labeled with these predicates. In contrast, a car can be nearby, at the corner, red, tall, and fixed, but not true.

Developmentally, Keil's key finding was that predicate hierarchies (predicability trees, as he called them) foliate via the refinement of existing classes of predicates, rather than by the radical reconstruction of predicate applicability relationships. This finding was supported by studies of the growth of predicability trees of young children. In other words, Keil found that predicability trees developed via the branching of the fringes of existing predicates -- the same process of refinement as was posited in the phonological model presented above. For instance, at the earliest ages studied (5-6 yrs.), some children's predicability trees looked like this:



(This is a single individual's tree.)

When second graders were tested, their trees were foliated versions of initial trees of this kind:



VALIDATING A THEORY OF HUMAN SKILL ACQUISITION

Kurt VanLehn
Xerox Parc
3333 Coyote Hill Road
Palo Alto, CA 94304

Extended Abstract

It is widely held that ordinary natural language conversations are governed by tacit conventions, called conversational postulates or felicity conditions. Learning a procedural skill is also a communication act. The teacher communicates a procedure to the student over the course of several lessons. The central idea of the theory to be presented is that *there are specific felicity conditions that govern learning*. Five felicity conditions on procedural skill acquisition have been discovered. They constrain how the teacher forms lessons and how the student learns from lessons. Even when lessons are incomplete, in that they do not contain enough information to communicate a procedure, students learn from them in ways determined by the felicity conditions. Thus, the theory can explain certain kinds of mis-learning as well as successful learning.

The theory employs a learning model. The model is a large AI-based computer program, named Sierra. It inputs a sequence of lessons consisting of worked examples and practice exercises. Sierra traverses the lesson sequence, learning a set of procedures (one procedure for each possible setting of certain parameters of individual difference). This set is very important. A key criterion for the theory is that the set of model-generated procedures should exactly match the set of procedures that students actually acquire, including their incorrect, "buggy" procedures. The model's predictions have been compared to data from students learning ordinary mathematical procedures: subtracting multi-digit numbers (1138 students), adding fractions (500 students) and solving simple algebraic equations (27 students).

However, there is more to validating this theory, or any complex AI-based theory, than just testing its predictions. The theory is stated as a set of 32 principles, of which the five felicity conditions are a subset. Sierra is only a tool, albeit an indispensable one, for calculating the predictions made by the principles. Explicating the principles is a necessary first step in validation. But many other questions remain to be addressed: What are the relationships between the principles and the behavior? Could the given cognition be simulated if the principles were violated or replaced by somewhat different ones? Would such a change produce inconsistency, or a plausible but as yet unobserved human behavior, or merely a minor perturbation in the predictions? Which alternatives if any can now be rejected in favor of the chosen principles? Answering these kinds of questions for each of the 32 principles has been the major work in validating the theory.

As an illustration of this kind of validation, consider the following felicity condition:

Learning a lesson introduces at most one new disjunction into a procedure.

In procedures, a disjunction may take many forms, e.g., a conditional branch (if-then-else). This felicity condition asserts that learners will only learn a conditional if each branch of the conditional is taught in a separate lesson—i.e.,

the then-part in one lesson, and the else-part in another.

The argument for the felicity condition hinges on an independently motivated principle: mathematical procedures are learned inductively. They are generalized from examples. There is an important philosophical-logical theorem concerning induction: If a generalization (a procedure, in this case) is allowed to have arbitrarily many disjunctions, then an inductive learner can identify which generalization it is being taught only if it is given *all possible examples*, both positive and negative. This is physically impossible in most interesting domains, including this one. If inductive learning is to bear even a remote resemblance to human learning, disjunctions must be constrained.

Two classic methods of constraining disjunctions are (i) to bar disjunctions from generalizations, and (ii) to bias the learner in favor of generalizations with the fewest disjunctions. The felicity condition is a third, new method. It uses extra input information, the lesson boundaries, to control disjunction. These three methods of controlling disjunctions are competing hypotheses for explaining how human learners control disjunction (along with several other hypotheses that won't be mentioned here).

The validation task is to discover and evaluate the entailments of each of the three hypotheses. In the case of the first hypothesis, it can be shown that barring disjunctions from procedures forces the theory to make absurd assumptions about the student's initial set of concepts—the primitive concepts from which procedures are built. Rejecting the first hypothesis leaves the fewest-disjunctions hypothesis and the one-disjunction-per-lesson hypothesis. The empirical predictions of these two hypotheses are identical, given the lesson sequences that occur in the data. More subtle arguments are needed to differentiate between them. Here are two.

(1) The felicity condition explains why lessons are so universally used in teaching and more importantly, why lesson sequences have the structure that they do. If the fewest-disjunctions hypothesis were true, then it would simply be an accident that lesson boundaries happened to fall exactly where disjunctions were being introduced. The one-disjunction-per-lesson hypothesis has more explanatory adequacy.

(2) The fewest-disjunctions hypothesis predicts that students would learn equally well if the example sequences were scrambled so that lesson boundaries fell at arbitrary points. This empirical prediction needs checking. If it is false, as I am sure it is, then the fewest-disjunctions hypothesis can be rejected on empirical as well as explanatory grounds.

This brief argument sketches the kind of individual support that each of the 32 principles has been given. Such competitive argumentation seems essential for demonstrating the psychological validity of a theory of this complexity.

procedure as one of fixing new rules of analysis, one automatically orders hypotheses so that the default assumption is that a rule is obligatory until proven otherwise. This ordering satisfies the Subset Principle.

The same point holds for determining the argument structure of verbs, such as *walk*: by default, the learner must hypothesize the narrowest possible surface language. Thus if an argument appears with a verb, such as, *John walked to the store*, then the default must be to take the Prepositional phrase *to the store* as an obligatory argument until proven otherwise by explicit positive evidence (such as a sentence where the argument does not appear, as in, *John walked*). Optionality must always be fixed positive examples.

IV. ACKNOWLEDGMENTS

This paper describes work done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Office of Naval Research under Office of Naval Research contract N00014-80-C-0505. The author would also like to thank Patrick Winston and Noam Chomsky for discussions about various aspects of this work.

V. REFERENCES

- Angluin, D., "Inductive Inference of Formal Languages from Positive Data," Information and Control, vol. 45, pp.117-135, 1980.
- Baker, C.L., "Syntactic Theory and the Projection Problem," Linguistic Inquiry, vol. 10, pp. 533-581, 1979.
- Chomsky, N., Rules and Representations, Columbia, New York, 1980.
- Chomsky, N. and Halle, M., The Sound Pattern of English, Harper and Row, New York, 1968.
- Gold, E.M., "Language Identification in the Limit," Information and Control, vol. 16, pp. 447-474, 1967.
- Kean, M.L., The Theory of Markedness in Generative Grammar, PhD dissertation, MIT Department of Linguistics, 1974.
- Keil, F., Semantic and Conceptual Development: an Ontological Perspective, Harvard, Cambridge, MA, 1979.
- Rizzi, L., "A Restructuring Rules in Italian Syntax," Recent Transformational Studies in European Languages, S.J. Keyser (Ed.), MIT Press, Cambridge, MA, pp. 113-158, 1978.
- Sommers, F., "Structural Ontology," Philosophia, vol. 1, pp.79-85, 1971.
- Williams, E., "Language Acquisition, Markedness, and Phrase Structure," Language Acquisition and Linguistic Theory, S. Tavakolian (Ed.), MIT Press, Cambridge, MA, pp.8-34, 1981.