KNOWLEDGE ACQUISITION FROM STRUCTURAL DESCRIPTIONS

Frederick Hayes-Roth and John McDermott
Computer Science Department[1]
Carnegie-Mellon University
Pittsburgh, Pa. 15213

February 11, 1976

ABSTRACT

The representation of concepts and antecedent-consequent productions is discussed and a method for inducing knowledge by abstracting such representations from a sequence of training examples is described. The proposed learning method, interference matching, induces abstractions by finding relational properties common to two or more exemplars. Three tasks solved by a program which performs an interference matching algorithm are presented. Several problems concerning the relational representation of examples and the induction of knowledge by interference matching are also discussed.

1 This research was supported in part by the Defense Advanced Research Projects Agency under contract no. F44620-73-C-0074 and monitored by the Air Force Office of Scientific Research.

I. INTRODUCTION

A number of distinct paradigms for studying learning machines have emerged during the last twenty years. Though each differs from the others in a variety of ways, the three differences which most clearly demark each paradigm are (1) the types of knowledge which can be acquired, (2) the way in which this knowledge is represented, and (3) the type of learning algorithm used. The learning machine which we will describe in this paper acquires concepts representable as conjunctive forms of the predicate calculus and behaviors representable as productions (antecedent-consequent pairs of such conjunctive forms); these concepts and behavior rules are inferred from sequentially presented pairs of examples by an algorithm that is provably effective for a wide variety of problems.

Learning is viewed here as a continual process of knowledge expansion, that is, as the acquisition, in adaption to training experiences, of higher-order, more complex, and more elaborate knowledge structures. One's knowledge at any point in time includes those concepts and productions innately provided or previously learned. The concepts are pattern templates; events which match a concept are recognized as belonging to the class delimited by that concept. The productions are pairs of concepts; one of the concepts functions as a recognizer, the other specifies the form of an associated action. A production is interpreted as a behavior generator in the sense that (in some computing environment with an appropriate control structure) the detection of a condition in the environment which matches the antecedent causes the consequent component to be instantiated and then evoked. Here both the antecedent and the consequent are templates; the antecedent determines whether the production is to be executed, and if so, what specific constants in the description of the event being attended to are to be bound to variables in the consequent.

*** Figure 1 goes about here

Within this framework, the machine learning problem with which we are concerned can be stated in the following way: Given a collection of concepts and productions constituting what is known at some time and a way of describing events in terms of their structure, construct a machine which is able to induce additional concepts or productions from training data. To make our treatment of this problem more concrete, we will use the simplest of the concept formation tasks attempted by our machine as an example throughout the paper. The task is to find what the three exemplars in Figure 1 have in common. Our program induces the following abstraction:

> There are three objects, including a small circle and a small square. The square is above the circle. The third object is large.

This paper is divided into six sections. In the next section we discuss in general a way of describing events which facilitates finding what two or more events have in common and a matching algorithm which can be used to find these abstractions. Then we locate SPROUTER, our concept and production inducing program, within the broader context of our work. The third section describes SPROUTER's interference matching (induction) algorithm in some detail; we indicate here more specifically how SPROUTER makes use of structural representations of events to acquire and store knowledge. In the fourth section we present the results of two concept formation tasks and one production inducing task, and in the fifth section we discuss some of the representational issues which our results help make evident. In the sixth section we conclude with a brief consideration of the strengths and weaknesses of SPROUTER.

II. STRUCTURAL REPRESENTATIONS AND INTERFERENCE MATCHING

The problem which we are addressing is simply described: Design a program which can infer concepts and productions from illustrative instances. The method we employ is correspondingly straightforward: Extract commonalities from the examples and attenuate their differences. Such an approach is like Galton's very primitive "composite photograph theory" of concept learning [3] and the "positive focusing strategy" for conjunctive concept learning first studied by Bruner, et al. [2]. While Galton's contribution was simply to propose that unknown patterns could be inferred by overlaying homologous memory representations of related examples (as if one were forming a composite of many photographs of the same subject), Bruner and his colleagues showed how such a process could in fact be realized. Each presented object (exemplar) is described as a conjunction of specific feature values.

To find the template which is matched by all of the presented objects, a feature vector containing only those features common to all of the exemplars is generated. This feature vector is the concept. Since that seminal work, many computer scientists have produced increasingly practical and sophisticated feature-value concept learners based on related techniques [6, 12, 13, 18].

Extending such learning models so that they can induce general (relational) classification and behavior rules is the goal of our work. In focusing on methods for generating relational abstractions which make possible the recognition of complex events, we encounter three problems not encountered in previous work. First we must develop a formal scheme for <u>describing</u> complex events which facilitates the generation of abstractions. Second, given descriptions of two examples of the same concept or production, we must develop a method for <u>comparing</u> them so that their commonalities can be identified. Third, it is necessary to develop a way of <u>storing</u> the discovered abstractions to facilitate their subsequent use in either of two ways: they may be used as templates for classification and behavior generation, or they may be used as knowledge representations whose precision may later be improved by learning if new instances of the same concept or production are provided. These problems are referred to below as the <u>description problem</u>, the <u>comparison problem</u>, and the <u>storage problem</u>. Each is considered in more detail in the subsequent paragraphs.

The description problem entails providing a symbolic representation of each exemplar which satisfies two demands. First, those attributes of the exemplar which are salient and potentially criterial must be reflected in its description to insure that the classification rule induced will be sufficiently discriminating. Note that since an exemplar may be composed of many objects, the description must distinguish each object and indicate clearly how it relates to the others. Second, the descriptions should facilitate the identification of commonalities among the exemplars so that the abstraction being sought can be found quickly. Since each object may exhibit a variety of characteristics and participate in numerous relationships with other objects, finding commonalities between two or more examples will necessitate search. A representational scheme which helps direct this search is almost essential.

The method of description we employ is built on three central concepts, the <u>property</u>, the <u>case frame</u>, and the <u>parameter</u>. A property is a feature or characteristic of an object. For example, SQUARE and SMALL name two properties of small squares; the properties ABOVE and BELOW are used in our work to describe objects which are above or below others in pictorial displays. To define the relationship of one object being above another, a case frame of the sort {ABOVE, BELOW} is used. In general, case frames are sets of properties which are semantically related in some exogenously determined manner. To produce descriptions of objects, events, or behaviors, case frames are parameterized (instantiated); that is, a name is given to each object in the event being described and this name is associated with each property of the object. Parameterized case frames are called <u>case relations</u>. For example, if b is the name of a square above a circle named c, this might be described by the following set of case relations: {{SQUARE: b}, {CIRCLE: c}, {ABOVE:b, BELOW: c}}. Such a set of case relations interpreted as a conjunction of valid propositions is called a parameterized structural representation or PSR [5, 8, 9]. In this example, {b, c} is the parameter set of the PSR.[1]

A structural description of the first two exemplars in the concept formation task discussed in the introduction is given below.

E1:
{{TRIANGLE:a, SQUARE:b, CIRCLE:c},
{LARGE:a, SMALL:b, SMALL:c},
{INNER:b, OUTER:a},
{ABOVE:a, ABOVE:b, BELOW:c},
{SAME!SIZE:b, SAME!SIZE:c}}

E2:
{{SQUARE:d, TRIANGLE:e, CIRCLE:f},
{SMALL:d, LARGE:e, SMALL:f},
{INNER:f, OUTER:e},
{ABOVE:d, BELOW:e, BELOW:f},
{SAME!SIZE:d, SAME!SIZE:f}}

The description of E1 asserts that there is an event composed of three objects, named a, b, and c; that the object labeled a has the properties of a triangle, of a

1 The PSR, as a description, corresponds exactly to an existentially quantified conjunction of predicates. In this example, the PSR is interpreted as $(\exists b,c)$ [SQUARE(b) $\wedge$ CIRCLE(c) $\wedge$ ABOVE(b,c)] with the appropriate interpretation for the three predicates. PSRs have proved to be more desirable bases for description than conventional predicate calculus formulae for numerous reasons: PSRs are easily written in compact forms embedding many case relations efficiently in a single set of property:parameter terms (each subset of such a compact relation instantiates any case frame comprising the same selection of properties); the interpretation of each argument (parameter) in a case relation is self-documented by the property name; and subsets of case relations are interpretable as abstractions of individual predicates. See Hayes-Roth [7].

large object, and of containing the object labeled b; and so on.

PSRs provide a solution to the storage problem as well as to the description problem; that is, they can be used in storing discovered abstractions. In the case of descriptions, parameter symbols are chosen to name each object so that if the same object is part of more than one case relation, it is referred to in a consistent way. If one alters the interpretation so that each distinct parameter is considered as an unbound variable, the PSR can be considered a template for concept identification. Such templates have been used by several researchers [1, 5, 8-10, 17] to specify what properties an object must have in order to satisfy membership in a pattern class. While the parameters in a description can be thought of as being existentially quantified, those in a PSR used as a template should be thought of as being universally quantified. When used as a template for pattern classification, the PSR is compared with an event (an existentially quantified PSR). If a mapping from the event to the template can be found which preserves the parameter bindings in the event description and which makes each case relation of the template true, the event is said to match the template.

In addition to their role as classification rules, PSRs can be used as general behavior rules. In this case two templates are associated. One of them, the antecedent, is used to recognize a set of conditions (a context) which indicates that a particular set of actions is appropriate; when the antecedent template is matched by some event in the environment, the rule is invoked. The second template, the consequent, specifies what actions are to be performed. When the two templates share common parameters, each parameter in the consequent is bound to the same value as the corresponding parameter in the antecedent. These behavior rules may act, for example, as Post productions, transformational grammar rules, or the problem solving rules of STRIPS [3]. In short, a rule with the antecedent A(X) and the consequent C(X) over the variables in the set X is interpreted to mean $(\forall X)\ [A(X) \Rightarrow C(X)]$. In actual applications, A defines a precondition which can be true of the contents of some working memory, and C defines what is to be done if the precondition is satisfied. Note that any such production can be described by a PSR in which each case relation in the antecedent includes a term of the sort EVENT:a, each case relation in the consequent includes a term of the sort EVENT:c, and the PSR itself includes a case relation {ANTECEDENT:a, CONSEQUENT:c}.

The abstraction of the first and second examples in the sample concept formation task can be represented as below.

E1*E2:

{{ABOVE:1,BELOW:2},
{SAME!SIZE:2,SAME!SIZE:1},
{SMALL:2},
{SQUARE:1},
{SMALL:1},
{CIRCLE:2},
{TRIANGLE:3},
{LARGE:3}}

Exemplar 1 is in fact an instance of this abstraction if the parameter 1 is replaced by the parameter b, the parameter 2 by c, and the parameter 3 by a. Likewise, exemplar 2 can be seen to match the abstraction if the parameter 1 is replaced by d, the parameter 2 by f, and the parameter 3 by e.

*** Figure 2 goes about here

The comparison problem can be solved by using a technique called <u>interference matching</u> or IM [7-8, 10]. It is a process for identifying all of the common properties of two PSRs and extracting a third PSR which is a template matched by the two exemplars. When two events have N attributes in common, their descriptions will contain at most N case relations which are identical (except for alphabetic differences between the names of corresponding parameters). Figure 2 schematizes IM as a process for finding the intersection containing these case relations. The circular areas labelled A and B correspond to two PSRs; all of the case relations common to the two PSRs are in the area labelled A*B (read "A star B"). Because any subset of this (conjunctive) set of common relations also defines an abstraction of A and B, it is important to be able to distinguish between the set and its proper subsets. We call any abstraction of A and B which is properly contained in no other abstraction of A and B a <u>maximal abstraction</u>. More formally, if S (*) A denotes that A is a PSR matched by the PSR S, then a maximal abstraction, A, of two PSRs, S and T, satisfies S(*)A and T(*)A and $(\forall B)$ $[B(*)A \wedge S(*)B \wedge T(*)B \Rightarrow A(*)B]$.

It should be pointed out that for any two PSRs, there may be more than one abstraction which is maximal in the above sense. For example, given the following two exemplars, E3: {{CIRCLE:a}, {RED:a},{LARGE:a}}

E4: {{CIRCLE:b}, {CIRCLE:c}, {RED:b}, {GREEN:c}, {SMALL:b}, {LARGE:c}}

two maximal abstractions exist. If the parameters a and b are considered to be identical, the maximal abstraction is

E3*E4: {{CIRCLE:1}, {RED: 1}}

If on the other hand, the parameters a and c are considered to be identical, the maximal abstraction is

E3*E4: {{CIRCLE:1}, {LARGE: 1}}

Thus in the language of PSRs, a maximal abstraction is defined to be the largest set of case relations that can be formed by intersection of the two compared sets of case relations when alphabetic differences between <u>bound</u> or <u>corresponding</u> parameters in the two PSRs are ignored. Parameter bindings may be defined by any one-one mapping between the parameter sets of the two PSRs. Note that an abstraction produced by assuming one particular set of parameter correspondences may be <u>submaximal</u>; that is, it may contain fewer relations than another abstraction which matches it but was produced by assuming a different parameter binding relation.

To perform interference matching on reasonably complex representations, we need an algorithm which, operating within as small a search space as possible, can discover the best maximal abstractions as quickly as possible. Two approaches to interference matching are known: (1) In the <u>bind-first</u> approach, each parameter in one PSR is associated with a parameter in the second PSR and then a maximal abstraction is found by extracting the case relations which are identical in the two PSRs (modulo the parameter bindings). In this case, if the lesser number of parameters (in either PSR) is MP and the greater number is NP, the number of possible binding functions is combinatorial, (binomial coefficient of NP over MP) * MP!. (2) Alternatively, in the <u>match-first</u> approach, all instantiations of case frames of one type in one PSR are compared with all instantiations of the same type of case frame in the other PSR, and possible parameter bindings are identified by determining which parameters have corresponding properties in comparable relations. Here if NI and MI are the numbers of case relations in the larger and smaller PSR (assuming only one type of case frame), the number of possible ways in which the relations can be forced into correspondence is similarly combinatorial. While it is true that if one were interested in computing abstractions of quite low-level event descriptions (such as undirected graphs) neither method would be much preferable to the other, in most real problems the number of instances of any particular case frame is quite small relative to the number of parameters in the PSR, and so the second method is usually preferable to the first. It is this method which is used in our current work.

The actual algorithm we use has the following form: A randomly selected case relation from one of the exemplar PSRs is put into correspondence with a case relation (which is a parameterization of the same case frame) from a second exemplar PSR; parameters having identical properties are identified as equivalent and the resulting common case relation becomes the (primitive) abstraction associated with that set of parameter bindings. Then other pairs of primitive case relations, one from each of the two exemplar PSRs, are put into correspondence. If a compared pair of relations entails parameter bindings consistent with those already identified, the common relation is added to the abstraction being produced. This new abstraction is the set union of the old abstraction and the new case relation, and the new set of parameter bindings is the set union of those bindings entailed by the previous abstraction and the forced bindings of the parameters in the compared pair of case relations. If a pair of case relations entails parameter bindings inconsistent with those already identified, the common case relation becomes a new (primitive) abstraction.

Clearly, this algorithm may find a number of competing maximal abstractions. Our approach is to build as many distinct abstractions as possible, one relation at a time, until a limitation on the number of distinct abstractions which can be considered at one time is exceeded. At that point, only those abstractions which are most significant in terms of the number and type of case relations they include are retained. These abstractions continue to be extended as other pairs of consistent relations are found; at the same time, the least significant abstractions are continually pruned from further consideration in order to keep the search space as small as possible.

The result of the process is a set of best maximal abstractions, represented as PSRs. Any one of these abstractions (interpreted as existentially quantified) can then be input to SPROUTER together with a third exemplar to produce a set of maximal abstractions of three exemplars, or the process may be repeated on as many additional exemplars as desired. Since a maximal abstraction is compared to an exemplar in the same way that an exemplar is compared to another exemplar, we find it desirable to store abstractions as PSRs, with the interpretation that their parameters represent existentially quantified variables derived from the correspondence of case relations in the exemplars from which the PSR was induced.

The successive steps involved in producing the maximal abstraction of the first two examples in the concept formation task are shown below.

(1) {SMALL:1}

(2) ({SMALL:1}, {ABOVE:2,BELOW:1})

(3) (({SMALL:1}, {ABOVE:2,BELOW:1}), {SAME!SIZE:1,SAME!SIZE:2})

(4) ((({SMALL:1}, {ABOVE:2,BELOW:1}), {SAME!SIZE:1,SAME!SIZE:2}), {SMALL:2})

(5) (((({SMALL:1}, {ABOVE:2,BELOW:1}), {SAME!SIZE:1,SAME!SIZE:2}), {SMALL:2}), {SQUARE:2})

(6) ((((({SMALL:1}, {ABOVE:2,BELOW:1}), {SAME!SIZE:1,SAME!SIZE:2}), {SMALL:2}), {SQUARE:2}), {CIRCLE:1})

The case relation {SMALL:c} is selected at random from E1 and is then put into correspondence with the case relation {SMALL:f} from E2. The parameters c and f are identified as equivalent and so (since c and f are the first pair of parameters bound) the primitive abstraction {{SMALL:1}} is generated. Then the pair of case relations {ABOVE:b, BELOW:c} and {ABOVE:d, BELOW:f} are put into correspondence. Since the identification of c with f and of b with d is consistent with the already established binding, the primitive abstraction {{ABOVE:2, BELOW:1}} is added to {{SMALL:1}}. It should be noted that our basic IM algorithm actually finds only six of the eight case relations constituting the abstraction. This is because the partial abstraction {{TRIANGLE:3}, {LARGE:3}} was pruned from consideration early in the match under the space limitation constraint. To insure that such complementary relations are not missed, our algorithm, after completing the process described above, searches for additional relations which can extend the abstractions produced. Any such relations which are found are conjoined to the abstraction to produce a maximal abstraction.

SPROUTER, the program which induces abstractions from structural descriptions, is only one part of a classification and learning system which we are developing. The top-level program, called SLIM [6], is a general space limited interference matching procedure which builds abstractions from examples and then uses these abstractions to classify test stimuli.[1] While the abstraction of feature-value repesentations can be performed by simple bit vector operations (which SLIM itself is capable of), the

1 Both SLIM and SPROUTER are implemented in SAIL for use on a PDP-10; SPROUTER loads in 14 thousand words of core.

generation of abstractions from PSRs requires the matching and parameter binding determinations discussed above. The program, SPROUTER, was created for this purpose. Once an abstraction is computed from some PSRs, it is nearly as complex a problem to use it for classification as it was to generate it originally. With this in mind, SPROUTER was designed to produce two outputs: one of these is a PSR, which as we have indicated can be matched with subsequent exemplars to produce more refined abstractions; the other is a special purpose recognition network used to exploit an abstraction as a template.

SLIM provides a general operating environment for concept (pattern) learning and classification. It is first given a set of exemplars all of which are known to belong to the same pattern class, and it induces abstractions (with the help of SPROUTER when necessary) by finding sets of common features or properties. This procedure can be repeated for different sets of exemplars until a number of abstractions have been built, each of which is an implicit rule for determining whether an event belongs to a particular pattern class. When SLIM is given an event to classify, its confidence in any particular classification judgment is determined by the abstraction's performance measure. This measure is a weighted combination of the a posteriori Bayesian probability of a correct classification less the probability of an incorrect classification. During the learning phase of processing, this measure is also used to eliminate insufficiently discriminating abstractions. By keeping the most discriminating abstractions, SLIM optimizes the expected overall performance of the limited set of templates it keeps as classifiers.

The templates which SPROUTER generates for SLIM are automatically compilable recognition networks or ACORNs [8, 9]. An ACORN is a special data structure, equivalent in representational power to a PSR, but better adapted to serve as a template; it is essentially a Pandemonium pattern recognition system [12], generalized to handle patterns and data described as general propositional formulae. Once an ACORN has been produced, SLIM can determine whether a descriptive PSR matches it by using the PSR to create an instance list at each of the lowest-level nodes in the ACORN and then allowing the relevant instances of subpatterns of interest to percolate upward in the network. If any instances of the highest-level node are found, the template is matched by the stimulus pattern. The lowest-level nodes of an ACORN correspond to the distinct case frames in a universally quantified PSR and are like the feature demons of a Pandemonium system. A feature demon, however, reports only the number of instances of its particular feature to higher-level demons, whereas the node in an ACORN actually passes its instances up to the higher-level nodes which it supports. The higher-level nodes look for instances of the particular conjunction of case relations in which they are interested, just as higher-level "cognitive demons" in Pandemonium look for specific combinations of feature values. The highest-level node in an ACORN is instantiated if and only if the abstraction is matched by the PSR. Thus this highest-level node corresponds to a Pandemonium's highest-level cognitive demon which recognizes when a pattern of interest is matched. Because ACORNs have been developed to provide a means for sharing the results of the evaluation of subexpressions common to numerous templates, each conjunction of predicates or subtemplates is associated with a single binary-branching node whose two descendants represent the conjoined propositional formulae.

Once a set of best maximal abstractions is computed for two or more exemplars, all training examplars (or a sample of them) may be examined to see if they match the inferred hypothetical concept or rule. Only to the extent that exemplars of the same class match an abstraction and those of the other classes do not, do we find support for the inference that the abstraction is the criterial concept underlying the training data [5-6]. ACORNs greatly facilitate this examination process. One simply instantiates the terminal nodes of the ACORN whose highest nodes represent the abstractions of interest, and then iteratively computes all instances of each higher-level node from those pairs of instances of its subordinate nodes which satisfy criterial tests on their values. If any instances of the abstraction are produced, the training exemplar matches the abstraction. Without ACORNs, it would be extremely difficult to determine which positive and negative training exemplars matched each abstraction.

A second reason for using ACORNs rather than some other sort of intermediate data structure is that only one generic representation of any abstraction need be computed during the search for maximal abstractions. Since each abstraction is associated with a node in an ACORN, equivalent abstractions can be easily identified and pruned from memory. This is done by computing all instances of each abstraction of the two exemplar PSRs and storing these at the associated ACORN node. If two instances of two different higher-level nodes are produced by conjunctions of identical sets of instances of the terminal nodes, the higher-level nodes represent equivalent abstractions and one may be deleted. Equivalently, we can recognize automorphic substructures of the compared PSRs whenever we find that the tests for one abstraction are satisfied by exactly the same case relations as the tests for some other abstraction. As will be shown later, since the tests on ACORN nodes completely specify the underlying PSR, the only way two nodes' tests can be satisfied by identical case relations is if the two nodes represent equivalent logical structures. Thus, ACORNs provide a basis for overcoming a difficulty which invariably arises with string type representations of PSRs (or equivalent predicate formulae) because many alphabetically distinct abstractions can be equivalent (each can match the other). For example, one may induce from examples the following abstraction for the concept triangle: Three vertices connected by three lines. Because there are three factorial distinct parameter binding relations between the vertices of one triangle and those of another, there are 6 binding functions and related case relation correspondences which entail equivalent abstractions. If each distinct abstraction of two PSRs were repesented only by a symbolic string, there would be no efficient way to determine that all of these alternative descriptions were identical. ACORNs facilitate this determination. Each ACORN node repesents a distinct PSR, and consequently equivalent PSRs are recorded as distinct instances of the same node in the network.

*** Figure 3 goes about here

Figure 3 shows the ACORN that is produced by SPROUTER for the first two exemplars of the concept formation task. Each of the nodes, (1) through (6), in the network corresponds to one of the partial abstractions given in the step-by-step derivation shown earlier. Nodes (7) and (8) are produced when the ACORN is extended. Note that if this ACORN were used to determine whether the third exemplar in the concept formation task is an instance of the class defined by the first two exemplars, SLIM would find that it is not since the large object in the third exemplar is not a triangle.

III. THE INTERFERENCE MATCHING ALGORITHM

SPROUTER's function, as we have said, is to build ACORNs which can be used by SLIM for recognition. Before this construction process can begin, a set of primitive (bottom-level) nodes must be generated and then instantiated. To generate these nodes, SPROUTER reads in the set of case frames which are relevant to the task it is facing. For each of these case frames, a primitive node is created which is essentially a universally quantified case relation. SPROUTER then finds, in the descriptive PSRs of two exemplars, the set of distinct instances (case relations) which are instances of each of these nodes. Each node has two associated instance lists; each of these lists contains the instances of the case relation for one of the exemplars. For example, given the two case frames N1: {CIRCLE}, N2: {ABOVE, BELOW} and the two exemplars

E5:	{{CIRCLE:a, CIRCLE:b}, {ABOVE:a, BELOW:b}}	E6:	{{CIRCLE:c}}

SPROUTER will create two nodes, N1 and N2, and then produce four instance lists. Two of these lists, ([E5/a], [E5/b]) and ([E6/c]), are associated with node N1. The other two, ([E5/a, E5/b]) and ( ), are associated with node N2.

When the primitive nodes have been instantiated, SPROUTER produces the set of maximal abstractions of the two PSRs by constructing, bottom-up, a binary-branching ACORN. Each higher-level node of this network is a conjunction of two nodes, one of which is always a primitive node. Before initiating the building process, SPROUTER deletes all of the primitive nodes which do not have at least one instance from each exemplar. Then one exemplar, the one with fewer instances over the remaining nodes, is tagged E_{intro}; the other exemplar is tagged E_{comp}. And each instance of E_{intro} is marked as <u>unused</u>. SPROUTER then begins the actual construction. An unused E_{intro} instance from a primitive node is chosen as one of the two instances to be used in the construction; it is selected on the basis of the likelihood of its being an instance of a node which is a constituent of a best maximal abstraction. This instance is then paired with every instance from E_{intro} of every node. Each of these pairs of instances is used to construct a candidate node which will accept instance pairs only if they are equivalent to the prototypic pair. If there is at least one such pair of instances in E_{comp}, the candidate node is added to the network and all instances of the node (from both exemplars) are computed. Thus, each step in the abstraction building process involves combining, iteratively, an unused instance from a primitive node with each other instance in the ACORN. After each of the resulting conjunctive nodes is generated for a pair of instances from E_{intro}, all instances of that node, first from E_{comp} and then from E_{intro}, are computed. If no instances are found in E_{comp}, the node represents an abstraction which is not true of the second exemplar and so the node is not added to the network. The process continues until all of the case relations that are common to both exemplars have been conjoined.

Of course, this algorithm, left unconstrained, would build a node for each subset of case relations in E_{intro} for which there was an equivalent subset in E_{comp}. Clearly, the size of the search space would increase exponentially. Thus, for even small problems, it is important to somehow reduce the number of nodes which are constructed. We use two heuristics. The first of these enables us to keep the search space to a manageable size by providing for the automatic pruning of those conjunctions which are least likely to be part of a best maximal abstraction. To determine which partial abstractions are least promising, a value is computed which we call the utility of a node. Basically, the utility of a node is an increasing function of the number of properties covered by the node and a decreasing function of the number of distinct parameters needed to instantiate the node. More specifically, our current utility measure adds 1.0 for each property of a case relation and subtracts 1.0 for each distinct parameter in the associated PSR. Our justification for this rather rough measure of utility is that it will yield as the highest valued nodes, those with the greatest scope and connectivity. Equivalently, the higher the utility of a node, the more informative and apparently "better" it is as an abstraction.

During the construction of the ACORN, a list of all nodes currently in the network is maintained. This list, which is ordered by the utility of its elements, has a stipulated maximum length. Whenever the number of total nodes in the ACORN exceeds this stipulated maximum, a primitive node which does not support any higher-order nodes is marked as removed from consideration. If all remaining primitive nodes support some higher-level node, then the least valued maximal abstraction (provided there is more than one maximal abstraction in the network) and all nodes suporting it (or supporting one of its supports, recursively) and not supporting some other higher valued maximal abstraction are deleted (or marked as removed from consideration if they are primitive nodes). Thus, the number of nodes in the network can exceed the stipulated maximum only if just one maximal abstraction remains. While in some cases, it might be desirable to require that at least k (k>1) best maximal abstractions be maintained, we have not yet found a need for this option.

As a result of the limitation on nodes in the ACORN, the typical behavior during construction is as follows: Instances are introduced one-at-a-time from E_{intro} and are conjoined with other E_{intro} node instances to form PSRs representing subsets of case relations of varying utility. As soon as the number of nodes corresponding to these nodes in the ACORN exceeds the stipulated maximum, the maximal node with the lowest utility together with all nodes which support only it are deleted from the network. This construction-and-pruning cycle is repeated until the set of best maximal abstractions has been found.

The second heuristic provides the search with direction by indicating which one of the unused instances is to be used in the next cycle of construction. Our search for the best maximal abstractions is essentially hill climbing, but occurs on many hills simultaneously. Since our pruning heuristic enables us to maintain a gradually decreasing number of maximal abstractions, the number of hills under consideration is reduced as the search progresses. Clearly, if we could select first all of those instances from E_{intro} which were instances of the best maximal abstractions (the highest hills), then our search, since it would take place in an essentially unimodal space, would be as efficient as possible. Of course it is impossible to determine a priori which instances are instances of the best maximal abstractions. However, by using a variant of the utility function described above, it is possible to compute, fairly cheaply, the upper bound of the actual utility of any node which might be constructed. Using this strategy, we can, at relatively little cost, significantly increase the probability that the node constructed will be a constituent of a best maximal abstraction. The selection procedure we use is as follows: We set a sampling factor (currently 20%) for the proportion of the unused instances from E_{intro} which are to be examined. We select at random this percent of the unused instances (but at least three until there are fewer than three unused instances). For each of the instances in this sample, we determine an upper bound of the utility of all of the nodes which could be constructed by conjoining the sampled instance with the remaining instances of nodes still under consideration. The one instance which produces the node with the highest potential utility is constructed.

The actual construction of a node is a two step process. First SPROUTER creates a set of tests which are both necessary and sufficient to accept just those instances which are equivalent to the pair of instances used as a model in building the higher-level candidate node. It is possible to create such a set of tests working only with the sameness or difference of selected parameters. For example, to construct an ACORN node to accept the two instances {CIRCLE:c} and {ABOVE:a, BELOW:c}, a <u>same parameter</u> (SP) test is generated to insure that the first parameter of the first case relation is the same as the second parameter of the second relation, and a <u>different parameter</u> (DP) test is generated to insure that no non-explicit SPs are accepted. If we think of this ACORN node as being constructed from a left and a right instance, where the parameter of the left instance is numbered 1, and the parameters of the right instance are numbered 2 and 3, then a minimally complete set of tests needed to exactly represent the same and different relations are {SP:1, SP:3} and {DP:1, DP:2}.

After the set of tests has been created, the candidate node is associated with a generator set which specifies how the parameters of its instances are to be extracted from pairs of subordinate instances which satisfy the node's SP and DP tests. Because of the implicit requirement for DP relations to hold on all distinct parameters, the order of the new relation is exactly the number of distinct parameters in the two relation instances used in building the node. In the above example, there would be two parameters in each instance of the new node and these would correspond to parameters 1 and 2 (since 1 and 3 are identical). The generator list for this node would be just (1,2). From the nature of the explicit SP and DP tests used, it follows that any two nodes having instances derived from equivalent pairs of instances must be equivalent. Whenever such a duplicate node is constructed, it is removed from the ACORN.

It should be apparent that an ACORN constructed in the fashion described above will not necessarily contain a maximal abstraction. Whether or not it will is partially dependent on what maximum has been stipulated for the number of nodes in the ACORN. But even if the stipulated maximum is large enough so that the highest node in the ACORN is a constituent of a maximal abstraction, the ACORN may not be complete; that is, some of the case relations in the abstraction may have been lost. This can occur if one or more primitive nodes whose instances are a part of the abstraction were removed from consideration early in the construction process. In such a case, however, it is always possible to extend the ACORN with conjunctions of these lost primitive node instances. This is done by successively re-introducing into the construct-and-prune cycle each instance in E_{intro} which does not support all of the instances of all of the highest nodes in the ACORN. Each re-introduced instance is conjoined with each of the instances of each highest node to produce candidate nodes. If instances of any of these new abstractions are found in E_{comp}, these new nodes are retained; the ACORN is then extended further, in the same way, until the best maximal abstractions have been found.

IV. THREE TASKS

In this section we will discuss SPROUTER's performance on three tasks. The first of these is just the simple concept formation task which we have been using as an example. The second task is a considerably more difficult concept formation problem. The third, the most difficult of the three, is a production inducing task; SPROUTER is given three pairs of sentences, each pair containing the active and passive version of the same sentence, and induces the general rule for transforming active sentences into passive ones. We have chosen these three tasks because each draws attention to an important dimension of SPROUTER's performance. The simple concept formation task shows SPROUTER's inability to deal with many-one parameter correspondences, a recently discovered problem of some importance that is discussed in the next section. The more complex concept formation task provides an example of the consequences of stipulating different values for the maximum number of abstractions that SPROUTER can retertain at any one time. Finally, the production learning task demonstrates that SPROUTER is powerful enough to find the best maximal abstractions in extremely large search spaces and, incidentally, that the IM algorithm is effective for inducing such rules of transformational grammar.

We have already seen the abstraction which SPROUTER constructs given the first two exemplars in the first concept formation task. The set of case frames from which the primitive nodes were created,[1] all three exemplars, and the best maximal abstraction found by SPROUTER are given below.

```
CF:
{N1:{CIRCLE},
 N2:{SQUARE},
 N3:{TRIANGLE},
 N4:{LARGE},
 N5:{SMALL},
 N6:{INNER, OUTER},
 N7:{ABOVE, BELOW},
 N8:{LEFT, RIGHT},
 N9:{SAME!SHAPE, SAME!SHAPE},
 N10:{SAME!SIZE, SAME!SIZE},
 N11:{BESIDE, BESIDE},
 N12:{CONTIGUOUS, CONTIGUOUS}}

E1:
```

1 This set, CF, was used for both concept formation tasks.

```
{{TRIANGLE:a, SQUARE:b, CIRCLE:c},
 {LARGE:a, SMALL:b, SMALL:c},
 {INNER:b, OUTER:a},
 {ABOVE:a, ABOVE:b, BELOW:c},
 {SAME!SIZE:b, SAME!SIZE:c}}

E2:
{{SQUARE:d, TRIANGLE:e, CIRCLE:f},
 {SMALL:d, LARGE:e, SMALL:f},
 {INNER:f, OUTER:e},
 {ABOVE:d, BELOW:e, BELOW:f},
 {SAME!SIZE:d, SAME!SIZE:f}}

E3:
{{SQUARE:g, CIRCLE:h, CIRCLE:i},
 {SMALL:g, LARGE:h, SMALL:i},
 {INNER:i, OUTER:h},
 {ABOVE:g, BELOW:h, BELOW:i},
 {SAME!SHAPE:h, SAME!SHAPE:i},
 {SAME!SIZE:g, SAME!SIZE:i}}

E1*E2*E3:
{{N10:{SAME!SIZE:1,SAME!SIZE:2}},
 {N7:{ABOVE:1,BELOW:2}},
 {N1:{CIRCLE:2}},
 {N5:{SMALL:1}},
 {N5:{SMALL:2}},
 {N2:{SQUARE:1}},
 {N4:{LARGE:3}}}
INSTANCES FROM EXEMPLAR E1*E2
([E1*E2/2,E1*E2/1,E1*E2/3])
INSTANCES FROM EXEMPLAR E3
([E3/g,E3/i,E3/h])
```

SPROUTER took 6 seconds of cpu time on a PDP-10 (model KA-10) to produce E1*E2 which it found after constructing 14 nodes (7 more than necessary). SPROUTER took 3 seconds and constructed 6 nodes (the fewest possible) to produce (E1*E2)*E3. The abstraction which SPROUTER found, however, though it is the best abstraction producible using our match-first method, is not maximal. It is missing two case relations. As we indicated in the first section of the paper, the abstraction which SPROUTER induces is the following:

> There are three objects, including a small circle and a small square.
> The square is above the circle. The third object is large.

The best maximal abstraction includes the specification that the large object contains another one which is one of the two small objects. SPROUTER is unable to find this abstraction for two reasons: (1) The grain size of the representations used in describing the examples is too big; more atomic uniform representations are needed to make abstraction, which is a subtractive process, more generally applicable. (2) Many-one parameter correspondences must be allowed in order to insure that relevant correspondences are not lost. These two problems, whose solution requires methods of greater generality than we have currently implemented, are discussed in detail in the next section. For the moment, the reader need know only that to produce a uniform PSR, every occurrence of the same parameter in the PSR is replaced by a distinct parameter and the several symbols referring to the same object are then related to one another by using the SP (same parameter) case frame {SP, SP}. The three exemplars in uniform PSR notation and the more complete abstraction which SPROUTER took a total of 5 minutes and 3 seconds to find are shown below.

E1:
{{TRIANGLE:a1, SQUARE:b1, CIRCLE:c1},
{LARGE:a2, SMALL:b2, SMALL:c2},
{INNER:b3, OUTER:a3},
{ABOVE:a4, ABOVE:b4, BELOW:c3},
{SAME!SIZE:b5, SAME!SIZE:c4},
{SP:a1, SP:a2, SP:a3, SP:a4},
{SP:b1, SP:b2, SP:b3, SP:b4, SP:b5},
{SP:c1, SP:c2, SP:c3, SP:c4}}

E2:
{{SQUARE:d1, TRIANGLE:e1, CIRCLE:f1},
{SMALL:d2, LARGE:e2, SMALL:f2},
{INNER:f3, OUTER:e3},
{ABOVE:d3, BELOW:e4, BELOW:f4},
{SAME!SIZE:d4, SAME!SIZE:f5},
{SP:d1, SP:d2, SP:d3, SP:d4},
{SP:e1, SP:e2, SP:e3, SP:e4},
{SP:f1, SP:f2, SP:f3, SP:f4, SP:f5}}

E3:
{{SQUARE:g1, CIRCLE:h1, CIRCLE:i1},
{SMALL:g2, LARGE:h2, SMALL:i2},
{INNER:i3, OUTER:h3},
{ABOVE:g3, BELOW:h4, BELOW:i4},
{SAME!SHAPE:h5, SAME!SHAPE:i5},
{SAME!SIZE:g4, SAME!SIZE:i6},
{SP:g1, SP:g2, SP:g3, SP:g4},
{SP:h1, SP:h2, SP:h3, SP:h4, SP:h5},

{SP:i1, SP:i2, SP:i3, SP:i4, SP:i5, SP:i6}}

```
E1*E2*E3:
{{N6:{INNER:1,OUTER:2}},
 {N0:{SP:3,SP:4}},
 {N7:{ABOVE:3,BELOW:5}},
 {N5:{SMALL:4}},
 {N0:{SP:6,SP:2}},
 {N4:{LARGE:6}},
 {N0:{SP:7,SP:2}},
 {N0:{SP:6,SP:7}},
 {N0:{SP:2,SP:8}},
 {N0:{SP:8,SP:6}},
 {N0:{SP:8,SP:7}},
 {N0:{SP:9,SP:5}},
 {N10:{SAME!SIZE:9,SAME!SIZE:10}},
 {N0:{SP:10,SP:4}},
 {N0:{SP:10,SP:3}},
 {N0:{SP:11,SP:9}},
 {N5:{SMALL:11}},
 {N0:{SP:12,SP:11}},
 {N0:{SP:9,SP:12}},
 {N1:{CIRCLE:12}},
 {N0:{SP:13,SP:10}},
 {N0:{SP:13,SP:4}},
 {N0:{SP:3,SP:13}},
 {N2:{SQUARE:13}}}
```

Though this abstraction includes the specification that the large object contains another object, it does not specify that this contained object is one of the two small objects. To induce that the contained object is small requires using a many-one parameter binding approach to interference matching discussed in the next section.

*** Figure 4 goes about here

The second concept formation task is significantly more complex than the previous one. Figure 4 displays the task. When SPROUTER was given this task and allowed a maximum of 9 nodes, it induced the following best maximal abstraction:

E1*E2*E3:

```
{{N10:{SAME!SIZE:1,SAME!SIZE:2}},
 {N7:{ABOVE:2,BELOW:1}},
 {N2:{SQUARE:1}},
 {N6:{INNER:3,OUTER:1}},
 {N5:{SMALL:3}},
 {N11:{BESIDE:1,BESIDE:2}},
 {N4:{LARGE:2}},
 {N7:{ABOVE:2,BELOW:3}},
 {N11:{BESIDE:3,BESIDE:2}},
 {N7:{ABOVE:4,BELOW:1}},
 {N9:{SAME!SHAPE:2,SAME!SHAPE:3}},
 {N4:{LARGE:1}},
 {N1:{CIRCLE:4}},
 {N10:{SAME!SIZE:4,SAME!SIZE:3}},
 {N5:{SMALL:4}},
 {N7:{ABOVE:4,BELOW:3}}}
INSTANCES FROM EXEMPLAR E1*E2
([E1*E2/1,E1*E2/2,E1*E2/3,E1*E2/4])
INSTANCES FROM EXEMPLAR E3
([E3/m,E3/j,E3/n,E3/l])
```

In other words:

> There are four objects. ehj(2) is the same shape as dgn(3) and is the same size as cfm(1). ehj(2) is above and beside both dgn(3) and cfm(1). dgn(3) is a small object and is contained in cfm(1) which is a large square. bil(4) is a small circle which is above both dgn(3) and cfm(1).

SPROUTER took 58 seconds to find E1*E2 and built 66 nodes. It took 47 seconds and built 52 nodes before finding (E1*E2)*E3, which is a conjunction of 16 nodes.

Given the same task, but with the constraint that the total number of nodes in the ACORN must not be greater than 8, SPROUTER produced the following abstraction:

```
E1*E2*E3:
{{N7:{ABOVE:1,BELOW:2}},
 {N7:{ABOVE:3,BELOW:2}},
 {N8:{LEFT:2,RIGHT:1}},
 {N11:{BESIDE:1,BESIDE:2}},
 {N10:{SAME!SIZE:3,SAME!SIZE:2}},
 {N10:{SAME!SIZE:4,SAME!SIZE:1}},
 {N9:{SAME!SHAPE:2,SAME!SHAPE:1}},
 {N7:{ABOVE:1,BELOW:4}},
 {N7:{ABOVE:3,BELOW:4}}}
INSTANCES FROM EXEMPLAR E1*E2
([E1*E2/2,E1*E2/3,E1*E2/4,E1*E2/1])
INSTANCES FROM EXEMPLAR E3
```

([E3/l,E3/k,E3/j,E3/n])

Though the stimpulated maximum for this run is only one less than the maximum of 9 stipulated for the previous run, the abstraction induced is very different:

> There are four objects. ehl(1) is the same shape as dgk(2) and is the same size as bij(3). ehl(1) is to the right of dgk(2). dgk(2) is the same size as cfn(4). ehl(1) and cfn(4) are above dgk(2) and bij(3).

This abstraction was sub-optimal because the stipulated node maximum was insufficient to allow SPROUTER to see beyond the seemingly promising LEFT, RIGHT relations.

The production inducing task is, of the three, by far the most difficult because the search space is so much larger and the abstraction so much more complex. SPROUTER was given the following three pairs of sentences:

(1) "The little man sang a lovely song." -->
"A lovely song was sung by the little man."
(2) "A girl hugged the motorcycles." -->
"The motorcycles were hugged by a girl."
(3) "People are stopping friendly policemen." -->
"Friendly policemen are being stopped by people."

*** Figure 5 goes about here

Figure 5 gives a graphical deep-structure representation of the first sentence. In PSR notation, this sentence is described by the following set of 64 case relations.

E1:
{{ANTECEDENT:e1, CONSEQUENT:e2},
{S:s1, NP:np11, VP:vp1, EVENT:e1},
{S:s2, NP:np21, VP:vp2, EVENT:e2},
{NP:np11, DET:the1, ADJ:little1, NOUN:noun11, EVENT:e1},
{NP:np21, DET:a1, ADJ:lovely1, NOUN:noun21, EVENT:e2},
{NOUN:noun11, NST:man1, NUMBER:n11, EVENT:e1},
{NOUN:noun21, NST:song1, NUMBER:n12, EVENT:e2},
{SINGULAR:n11, EVENT:e1},
{SINGULAR:n12, EVENT:e2},

{VP:vp1, AUX:aux11, VERB:verb11, NP:np22, EVENT:e1},
{SAME!NP:np21, SAME!NP:np22},
{NP:np22, DET:a2, ADJ:lovely2, NOUN:noun22, EVENT:e1},
{SAME!NOUN:noun21, SAME!NOUN:noun22},
{NOUN:noun22, NST:song2, NUMBER:n13, EVENT:e1},
{SINGULAR:n13, EVENT:e1},
{VP:vp2, AUX:aux12, PB:pb1, VERB:verb12, PP:pp1, EVENT:e2},
{AUX:aux11, AUXST:have1, TENSE:t11, NUMBER:n15, EVENT:e1},
{AUX:aux12, AUXST:have2, TENSE:t12, NUMBER:n16, EVENT:e2},
{SAME!AUX:aux11, SAME!AUX:aux12},
{VERB:verb11, VST:sing1, TENSE:t21, NUMBER:n15, EVENT:e1},
{VERB:verb12, VST:sing2, TENSE:t22, NUMBER:n16, EVENT:e2},
{SAME!VERB:verb11, SAME!VERB:verb12},
{PB:pb1, PBST:be1, TENSE:t23, NUMBER:n16, EVENT:e2},
{SAME!TENSE:t11, SAME!TENSE:t12},
{SAME!TENSE:t21, SAME!TENSE:t22, SAME!TENSE:t23},
{SINGULAR:n15, EVENT:e1},
{SINGULAR:n16, EVENT:e2},
{PRESENT:t11, EVENT:e1},
{PRESENT:t12, EVENT:e2},
{PAST-PART:t21, EVENT:e1},
{PAST-PART:t22, PAST-PART:t23, EVENT:e2},
{PP:pp1, PREP:by1, NP:np12, EVENT:e2},
{SAME!NP:np11, SAME!NP:np12},
{NP:np12, DET:the2, ADJ:little2, NOUN:noun12, EVENT:e2},
{SAME!NOUN:noun11, SAME!NOUN:noun12},
{NOUN:noun12, NST:man2, NUMBER:n14, EVENT:e2},
{SAME!NUMBER:n11, SAME!NUMBER:n12, SAME!NUMBER:n13,
SAME!NUMBER:n14, SAME!NUMBER:n15, SAME!NUMBER:n16},
{SINGULAR:n14, EVENT:e2},
{THE:the1, EVENT:e1},
{THE:the2, EVENT:e2},
{SAME!WORD:the1, SAME!WORD:the2},
{LITTLE:little1, EVENT:e1},
{LITTLE:little2, EVENT:e2},
{SAME!WORD:little1, SAME!WORD:little2},
{MAN:man1, EVENT:e1},
{MAN:man2, EVENT:e2},
{SAME!WORD:man1, SAME!WORD:man2},
{HAVE:have1, EVENT:e1},
{HAVE:have2, EVENT:e2},
{SAME!WORD:have1, SAME!WORD:have2},
{SING:sing1, EVENT:e1},
{SING:sing2, EVENT:e2},
{SAME!WORD:sing1, SAME!WORD:sing2},
{A:a1, EVENT:e1},
{A:a2, EVENT:e2},
{SAME!WORD:a1, SAME!WORD:a2},
{LOVELY:lovely1, EVENT:e1},
{LOVELY:lovely2, EVENT:e2},

{SAME!WORD:lovely1, SAME!WORD:lovely2},
{SONG:song1, EVENT:e1},
{SONG:song2, EVENT:e2},
{SAME!WORD:song1, SAME!WORD:song2},
{BE:be1, EVENT:e2},
{BY:by1, EVENT:e2}}

*** Figure 6 goes about here

The best maximal abstraction found by SPROUTER is illustrated in figure 6. The arrows in figure 6 indicate where the abstraction contains case relations representing that the connected nodes are the same part of speech (e.g., are both noun phrases, nouns, verb phrases, etc.) or have the same value (e.g., are both singular or both the same word). These case relations were provided for each training sentence as indicated in the preceding PSR for the sentence pair E1. Basically, these case relations connect two "tokens" of the same grammatical "type". Those relations that have survived the interference matching process can now be interpreted as identifying parameters in the antecedent and consequent events which should be considered identical. As previously explained, when the inferred production is used to produce behavior and a PSR in working memory matches the antecendent component of this rule, variable values will be bound and substitutions will be made into the consequent event as prescribed by the arrows. In an effort to simplify the figure, boxes have been constructed around any group of antecedent nodes where each contained parameter is connected by a "same" type relation to the corresponding parameter in the consequent box. SPROUTER took 19 minutes and 15 seconds and built 124 nodes in constructing E1*E2 and took 14 minutes and 33 seconds and built 97 nodes in constructing (E1*E2)*E3. Since the rule which it induced contains 45 distinct parameters over 40 case relations, we can take 45! as a lower bound on the size of the search space; that is, there are 45! (approximately 10^{57}) possible one-one parameter binding relations which could be established between any pair of parameter sets from E1, E2, or E3. SPROUTER made 81 bad decisions (constructed nodes which did not support the eventual maximal abstraction) in computing E1*E2 and 57 bad decisions in computing (E1*E2)*E3.

V. PROBLEMS IN REPRESENTATION AND MATCHING

As SPROUTER's performance on the first of the concept formation tasks shows, there are two problems which arise in the learning methodology that we have described. The first is that some learning problems can only be solved if the implicit semantics of the case frame structure are made explicit in more elaborate and primitive uniform representations. The second is that, even with uniform representations, some learning problems require the identification of many-one parameter correspondences in order to produce maximal abstractions and thus cannot be solved by SPROUTER or any other program using a one-one matching method. Each of these problems is discussed in turn.

The need for uniform representations can best be conveyed through a simple learning example. Suppose we have two examples of the concept "two line segments, connected in at most one place" whose descriptions are provided in terms of the binary symmetric case frame {ENDPOINT, ENDPOINT} identifying the two endpoints of a line segment. Let the two examples be E1: {{ENDPOINT:a, ENDPOINT:b}, {ENDPOINT:c, ENDPOINT:d}} and E2: {{ENDPOINT:w, ENDPOINT:x}, {ENDPOINT:x, ENDPOINT:y}}. E1 describes two disjoint lines and E2 describes two lines connected at vertex x. Implicit in these PSRs are the assumptions that two endpoints are the same if and only if they are labeled by the same parameter. In order to recognize that both E1 and E2 match a maximal abstraction which represents the concept to be learned (two lines whether or not connected at a common point), it is apparently necessary to establish parameter correspondences between two parameters in E1 (say b and c) and one parameter in E2 (say x). To avoid this necessity and to permit induction of the most informative abstractions, uniform PSRs are employed which make explicit the same parameter (SP) and different parameter (DP) relationships between each pair of parameters in a description.

While a detailed discussion of the formal characteristics of uniform representations occurs elsewhere [7, 10], several important properties will be pointed out here. First, rather than using one parameter (say p) in every case relation in which the same object is cited, uniform PSRs employ distinct symbols (e.g., p', p", ...) for each. To preserve the information that the various parameters all refer to the same object, every pair (e.g., p', p") of these parameters is used to instantiate an SP case frame, such as {SP:p', SP:p"}. Similarly, every pair of parameters (p',q') which refer to distinct objects in the PSR are used to instantiate a DP case frame, {DP:p', DP:q'}. If the preceding exemplars E1 and E2 are represented by uniform PSRs, the maximal abstraction which would be produced by SPROUTER would be E1*E2: {{ENDPOINT:1, ENDPOINT:2}, {ENDPOINT:3, ENDPOINT:4}, {DP:1, DP:2}, {DP:1, DP:3}, {DP:1, DP:4}, {DP:2, DP:4}}. This abstraction would be entailed by the parameter bindings 1=a=w, 2=b=x', 3=c=x", 4=d=y. The fact that the case relations {DP:b, DP:c} in E1 and {SP:x', SP:x"} in E2 did not match would simply be lost. The resulting abstraction E1*E2 would then be properly interpreted as meaning, "There are two lines, with endpoint pairs (1,2) and (3,4), such that all points are distinct except perhaps 2 and 3. Without uniform representations, SPROUTER's requirement for one-one parameter correspondences would have meant that the best abstraction that could have been produced would include only the one case relation {ENDPOINT:1, ENDPOINT:2}.

Furthermore, it can be seen that there are other induction problems which will not be solved correctly by SPROUTER's match-one-case-relation-at-a-time approach. Specifically, when abstractions entail discovering that only some parts of case relations of two PSRs match, the maximal abstraction should reflect the common subset of property:object terms. This can be accomplished if each case relation of the form {property$_1$:x_1, ..., {property$_n$:x_n} is replaced by the set of <u>uniform case relations</u> {{property$_1$:x_1}, ..., {property$_n$:x_n}, {SCR:x_1, SCR:x_2}, ..., {SCR:x_{n-1}, SCR:x_n}}, interpreted as follows. Each object x_i has some attribute property$_i$ and each pair of objects x_i, x_j ($1 \le i < j \le n$) occurred in the same case relation (SCR). As a result of this more atomic description of the case relation, abstractions including only a part of a PSR case relation will be reflected as the largest subset of the associated uniform case relations which is common to the two compared PSRs.

Because SPROUTER knows nothing about the semantics of its PSRs, learning tasks may be specified using PSRs whose case frames are at the highest level of description appropriate, which in some cases will be the atomic level of uniform PSRs. SPROUTER simply assumes that every pair of references to identical (different) parameters entails an SP (DP) test. Thus, the user of SPROUTER can choose the level of representation which is suitable for the learning problem to be solved. Because uniform PSRs include more case relations and parameters, abstractions based on them require more search and consequently more computing time. Thus, we use the uniform reprsentation only when necessary. As this discussion suggests, determining the appropriate grain for a representation seems not as much a formal question as a question of empirical sufficiency in particular induction task domains. Therefore, we see the aspect of our work concerned with finding the appropriate grain of representation for various problems as inherently experimental and empirical.

The second problem we encountered concerns the feasibility of abstraction methods based on one-one parameter binding functions. SPROUTER requires this type of binding and exploits this restriction to reduce the search space of possible solutions. If one thinks of PSRs as graph representations, where vertices correspond to parameters and edges to SP, DP, and SCR relations, it is possible to show that interference matching is equivalent to finding the common subgraphs of two event description graphs [7, 10]. In other words, the one-one parameter correspondence requirement is a restriction that each vertex in one event graph is permitted to match at most one vertex in the other graph. While this seems "formally" attractive, it is overly restrictive for a variety of learning tasks. For example, in order to find the best maximal abstraction in the first concept formation task, for each pair of examplars, the small object which is inside the large object in one of the exemplars must be permitted to match both small objects in the other exemplar. Though this problem is superficially similar to the grain size problem, the use of uniform PSRs with explicit SP and DP relations is inadequate to overcome it. The problem can be solved only by allowing many-one parameter correspondences and consequently requires more general methods than those currently developed. A very simple example can illustrate the general problem. Let E1 be {{SMALL:x}, {SQUARE:x}, {RED:x}} and E2 be {{SMALL:y}, {SQUARE:y}, {SQUARE:z}, {RED:z}}. In both examples, there is a small square and a red square, but there is only one square in E1 and there are two in E2. In order to produce the correct abstraction of E1 and E2, which in uniform representation is {{SMALL:1}, {SQUARE:2}, {SP:1, SP:2}, {SQUARE:3}, {RED:4}, {SP:3, SP:4}}, our method needs to be modified to allow the single instance of the SQUARE case frame in E1 to match two instances of it in E2. Because it is impossible to know a priori which case relations must be matched to more than one case relation in a compared PSR, it would be very difficult to modify the match-first IM algorithm to handle such problems even if many-one bindings were allowed.

The best solution we know of to this problem uses the bind-first approach to interference matching. The method can be described as follows: First, uniform PSRs E1' and E2' are generated to replace the exemplar PSRs E1 and E2. If the parameter sets of E1' and E2' are P and Q, where $|P|$ is less than or equal to $|Q|$, then each possible parameter binding relation for an abstraction is a set $B = \{(p,q) : p \in P, q \in Q\}$ where $(\forall\, p \in P, \forall\, q \in Q)\ (\exists\, p' \in P, \exists\, q' \in Q)\ (p,q') \in B \wedge (p',q) \in B \wedge |B| = |Q|$. In other words, each correspondence binding relation between the parameters of the uniform PSRs associates at least one parameter in E1' to each parameter in E2' (and vice versa) and establishes one correspondence for each of the parameterized references to objects in the other PSR. Of course, those binding relations which entail the identification of many commalities between E1' and E2' are the most preferred.

While it appears that this generalization of the one-one binding method will be infrequently needed, such a generalization now seems essential for the development of completely general learning machines. We are currently designing a many-one, bind-first interference matching program which can overcome the now apparent weaknesses of the one-one, match-first method.

At this point, it is desirable to relate our work to earlier reseach efforts. Similar, but less general, relational abstraction methods have been studied by Plotkin [14, 15], Vere [19], and Winston [20]. Weaknesses of the previous work which are considered here include the failure to utilize DP relations, a dependence upon restricted and exponential enumerative algorithms, and an assumption of the sufficiency of the one-one binding relation. Because all of the earlier researchers failed to realize the necessity for DP relations to force distinct value bindings for distinct variables, their learning algorithms would, for example, permit a single line segment to instantiate all three distinct line segment predicates in the triangle template, "three line segments, L1, L2, and L3, connected at their endpoints." Winston's learning methods were restricted to toy block construction problems using only unary and binary predicates such as adjacency of two blocks and are apparently not extensible to different domains. On the other hand, Plotkin and Vere studied the abstraction problem in terms of general n-ary predicates, but could infer concepts only corresponding to sets of (non-uniform) case relations and SP tests. While Hayes-Roth [7, 10] was the first to show formally that the IM algorithm could be used for inducing productions from antecedent-consequent training examples, our work is the first to demontrate its feasibility. The chief drawback of all of the previous work, however, was its reliance upon enumerative matching procedures. As we have tried to show, interference matching is best viewed as an exponential search problem which is, fortunately, apparently amenable to simple heuristic methods. Because IM is an NP-complete procedure (it subsumes the graph monomorphism problem), exhaustive procedures are simply not feasible for solving even moderately complex problems.

Interestingly, Hayes-Roth, Plotkin, and Vere each independently proved that their particular enumerative algorithms provided effective solutions to the "induction problem" which each of them had formalized in terms of various assumptions about what needed to be learned. All of these previous formalizations are inadequate to solve the type of learning problem introduced in this paper as necessitating many-one bindings. That is, all previous theoretical approaches assume the sufficiency for abstraction of the one-one parameter binding relation. As we have shown, however, with one simple example, any axiomatic system incorporating this assumption is inadequate as a general framework for representation and learning.

VI. CONCLUDING REMARKS

SPROUTER has already solved learning problems of theoretical significance and of considerable complexity. Because of the extensive size of the search spaces, such learning could not be done with simple enumerative matching algorithms. In essence, SPROUTER establishes the feasibility of induction from non-trivial exemplar descriptions. In many respects, however, SPROUTER is quite primitive. It is a purely syntactic matcher; it knows nothing at all about the underlying structure or significance of any of the predicate descriptions it operates upon. For this reason, its utility function, and thus its heuristics, are very weak. One interesting aproach to improving the performance of SPROUTER would be to provide it with domain-specific utility functions. For example, if SPROUTER knew that concordance on antecedent or consequent relations was more important than concordance on most other relations, it would never attempt to match the antecedent part of an example with a consequent part. Similarly, if it knew that concordance of higher-order grammatical constructs (e.g., a sentence) was more significant than concordance on lower-order ones, it could quickly zero in on the concordances of two sentence structures and then continue building abstractions in an essentially top-down fashion.

Even though SPROUTER's performance has been quite impressive on several tasks, there are a number of difficulties impeding the use of such a learning machine in general applications. First, an empirical question has been raised regarding the preferability of approaches to induction based on the one-one and many-one binding alternatives. If object integrity in representations is generally tenuous--that is, if each object in one PSR can correspond to multiple, diverse objects in another PSR, as was the case in the first concept formation task--abstraction procedures based on the many-one approach will have to be developed. Secondly, one must identify which real-world problems can be solved by interference matching methods. Because the case frames which SPROUTER uses in inferring abstractions are assumed to be externally provided, the utility of our method depends upon the prior identification of the criterial properties of events. Thus while SPROUTER can solve many concept learning and production inducing problems if it is provided the relevant case frames, it remains to be shown that this will be a sufficiently powerful basis for computer-based learning.

REFERENCES

1. Barrow, H. G., Ambler, A. P., and Burstall, R. M. Some techniques for recognizing structures in pictures. In Frontiers of Pattern Recognition, S. Watanabe (Ed.), Academic Press, New York, 1972.

2. Bruner, J. S., Goodnow, J. J., and Austin, G. A. A Study of Thinking. Wiley, New York, 1956.

3. Fikes, R. E., and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, (1971), 189-208.

4. Galton, F. Inquiries into Human Faculty and its Development. Dent, London, 1907.

5. Hayes-Roth, F. A structural approach to pattern learning and the acquisition of classificatory power. Proc. First Intl. Jt. Conf. Pattern Recognition, 1973.

6. Hayes-Roth, F. Schematic classification problems and their solution. Pattern Recognition 6, 2 (Oct. 1974), 105-114.

7. Hayes-Roth, F. Fundamental mechanisms of intelligent behavior: the representation, organization, acquisition, and use of structural knowledge in perception and cognition. Doctoral Dissertation, The University of Michigan, Ann Arbor, 1974.

8. Hayes-Roth, F. An optimal network representation and other mechanisms for the recognition of structured events. Proc. Second Intl. Jt. Conf. Pattern Recognition, 1974.

9. Hayes-Roth, F. Representation of structured events and efficient procedures for their recognition. Pattern Recognition (in press).

10. Hayes-Roth, F. Uniform representations of structured patterns and an algorithm for grammatical inference. Working Paper, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, 1975.

11. Hayes-Roth, F., and Mostow, D. J. An automatically compilable recognition network for structured patterns. Proc. Fourth Intl. Jt. Conf. Artificial Intelligence, 1975.

12. Hunt, E. B. Concept Formation: An Information Processing Problem. Wiley, New York, 1962.

13. Michalski, R. S. AQVAL/1--Computer implementation of a variable valued logic system VL_1 and examples of its application to pattern recognition. Proc. First Intl. Jt. Conf. Pattern Recognition, 1973.

14. Plotkin, G. D. A note on inductive generalization. In Machine Intelligence, vol. 5, B. Meltzer and D. Michie (Eds.), American Elsevier, New York, 1970.

15. Plotkin, G. D. A further note on inductive generalization. In Machine Intelligence, vol. 6, B. Meltzer and D. Michie (Eds.), American Elsevier, New York, 1971.

16. Selfridge, O. G. Pandemonium: a paradigm for learning. Symposium on the Mechanisation of Thought Processes. H. M. Stationery Office, 1959.

17. Shaw, A. C. Picture graphs, grammars, and parsing. In Frontiers of Pattern Recognition, S. Watanbe (Ed.), Academic Press, New York, 1972.

18. Stoffel, J. C. A classifier design technique for discrete variable pattern recognition problems. IEEE Trans. on Computers **C-23**, 4 (Apr. 1974), 428-441.

19. Vere, S. A. Induction of concepts in the predicate calculus. Proc. Fourth Intl. Jt. Conf. Artificial Intelligence, 1975.

20. Winston, P.H. Learning structural descriptions from examples. AI-TR-76. Cambridge: MIT Artificial Intelligence Laboratory, 1970.

FIGURE CAPTIONS

Figure 1. The first concept formation task.

Figure 2. Interference matching.

Figure 3. The ACORN for E1 * E2 in the first concept formation task.

Figure 4. The second concept formation task.

Figure 5. Example E1 for the production inducing task. The example comprises two sentences, the antecedent above and the consequent below.

Figure 6. The active-to-passive transformational grammar rule induced from 3 examples. Arrows indicate variable substitutions from the antecedent to the consequent components.

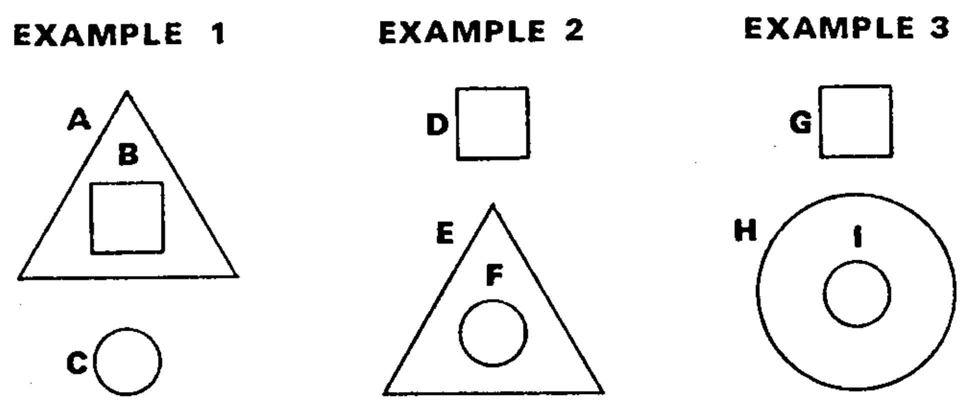

Figure 1. The first concept formation task.

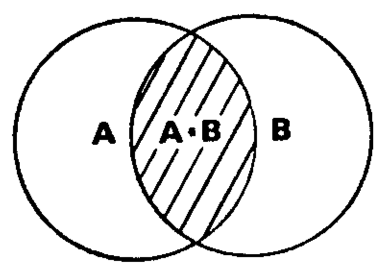

Figure 2. Interference matching.

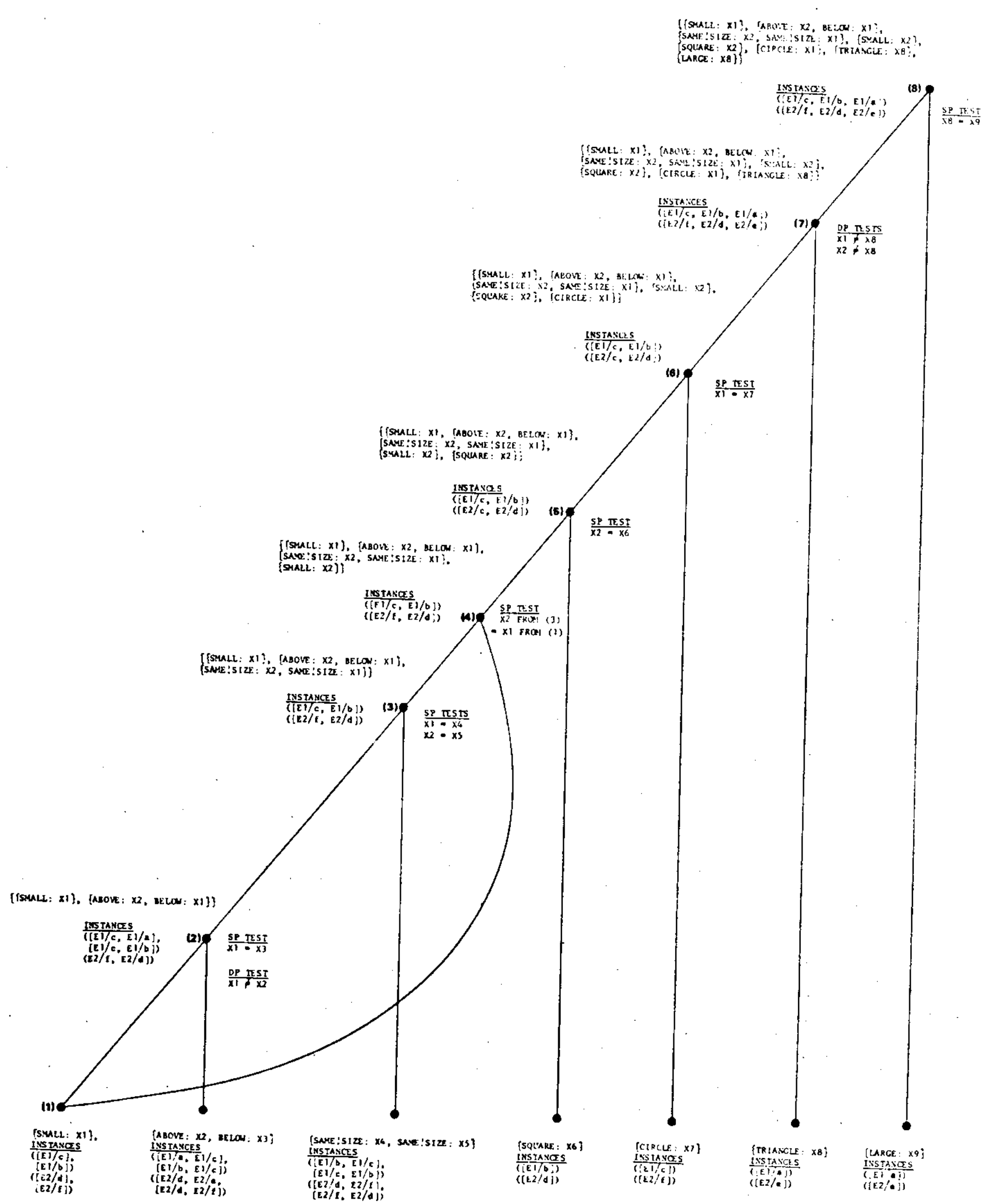

Figure 3. The ACORN for E1 * E2 in the first concept formation task.

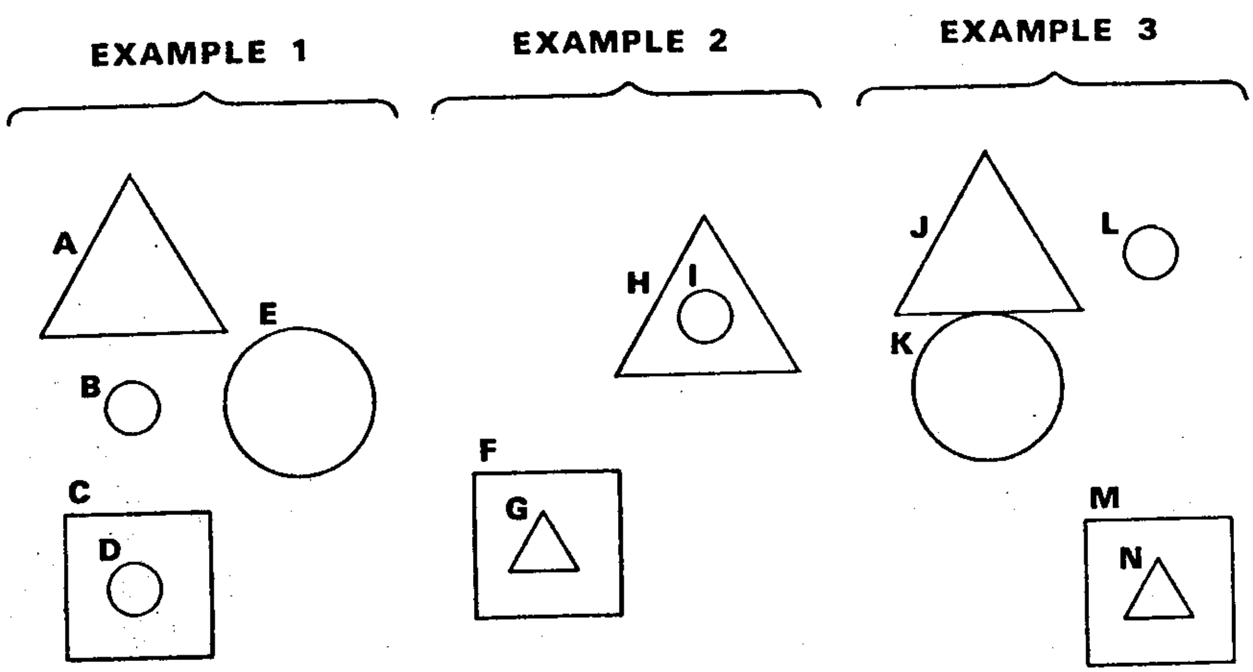

Figure 4. The second concept formation task.

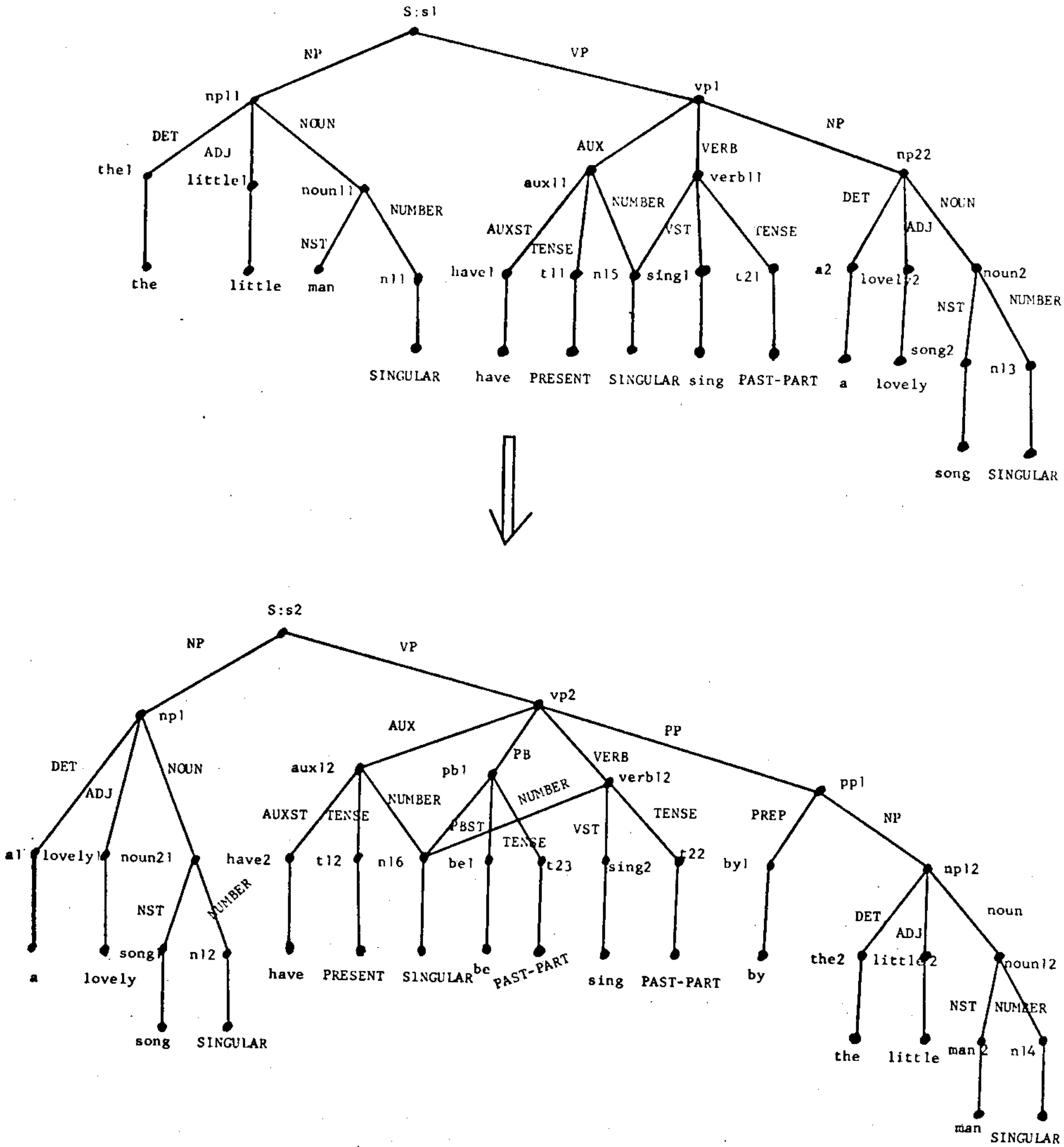

Figure 5. Example E1 for the production inducing task. The example comprises two sentences, the antecedent above and the consequent below.

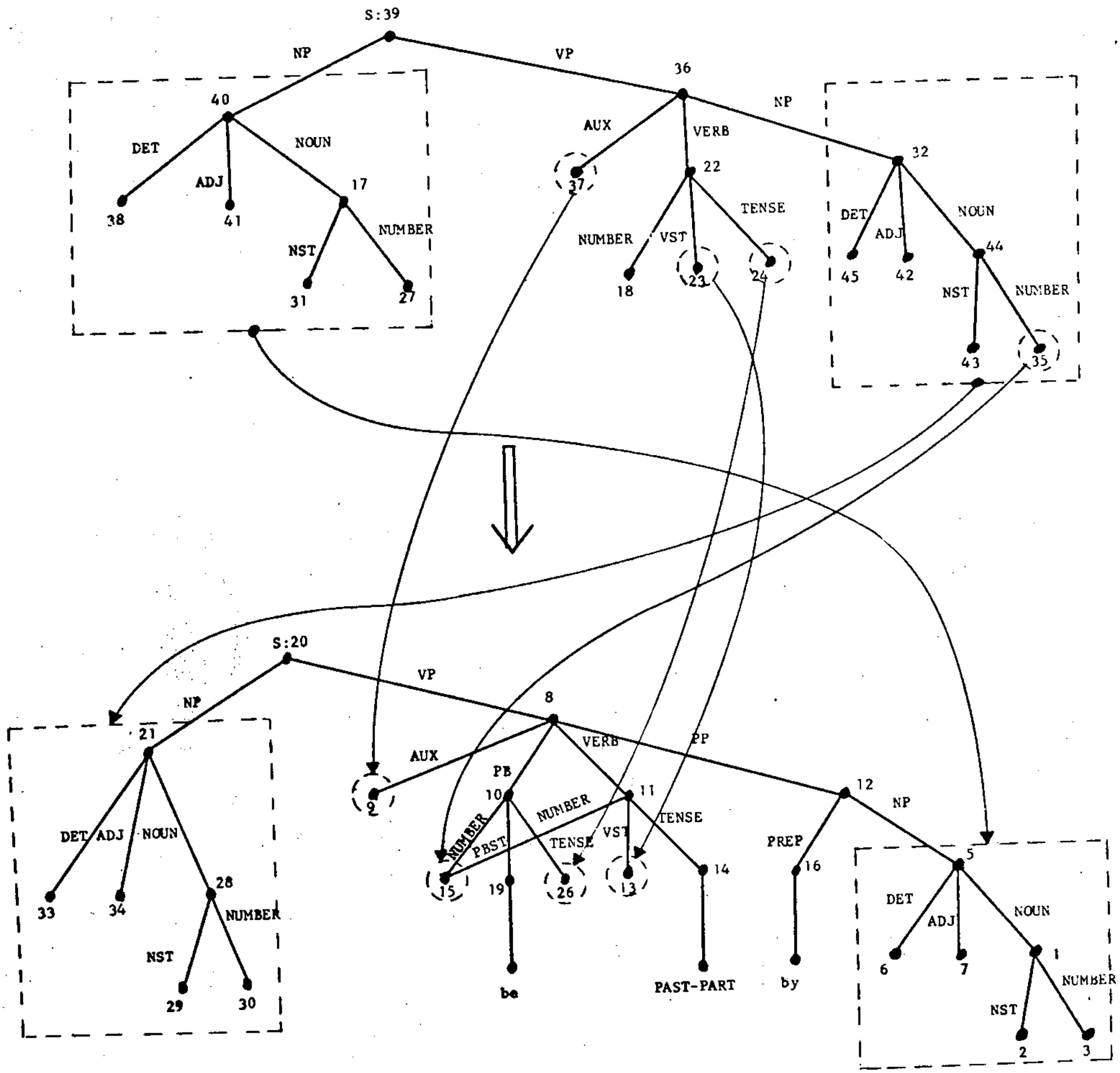

Figure 6. The active-to-passive transformational grammar rule induced from 3 examples. Arrows indicate variable substitutions from the antecedent to the consequent components.