MACHINE LEARNING

An Artificial Intelligence Approach

Contributing authors:

John Anderson
Ranan Banerji
Gary Bradshaw
Jaime Carbonell
Thomas Dietterich
Norman Haas
Frederick Hayes-Roth
Gary Hendrix
Patrick Langley
Douglas Lenat
Ryszard Michalski
Tom Mitchell
Jack Mostow
Bernard Nudel
Michael Rychener
Ross Quinlan
Herbert Simon
Derek Sleeman
Robert Stepp
Paul Utgoff

Editors:

Ryszard S. Michalski
University of Illinois
at Urbana-Champaign, IL

Jaime G. Carbonell
Carnegie-Mellon University
Pittsburgh, PA

Tom M. Mitchell
Rutgers University
New Brunswick, NJ

tioga publishing company
palo alto, california

3

A COMPARATIVE REVIEW OF SELECTED METHODS FOR LEARNING FROM EXAMPLES

Thomas G. Dietterich
Stanford University

Ryszard S. Michalski
University of Illinois at Urbana-Champaign

ABSTRACT

Research in the area of learning structural descriptions from examples is reviewed, giving primary attention to methods of learning characteristic descriptions of single concepts. In particular, we examine methods for finding the maximally-specific conjunctive generalizations (MSC-generalizations) that cover all of the training examples of a given concept. Various important aspects of structural learning in general are examined, and several criteria for evaluating structural learning methods are presented. Briefly, these criteria include *(i)* adequacy of the representation language, *(ii)* generalization rules employed, *(iii)* computational efficiency, and *(iv)* flexibility and extensibility. Selected learning methods developed by Buchanan, *et al.*, Hayes-Roth, Vere, Winston, and the authors are analyzed according to these criteria. Finally, some goals are suggested for future research.

3.1 INTRODUCTION

3.1.1 Motivation and Scope of Chapter

The purpose of this chapter is to introduce some of the important issues affecting the design of learning programs—particularly programs that learn from examples. This chapter begins with a survey of these issues. From the survey,

four criteria are developed for evaluating learning methods. The remainder of the chapter describes and evaluates five existing learning systems according to these criteria.

We do not attempt to review all of the work on learning from examples (also known as *learning by induction*). Instead, we focus on one particular problem: the problem of learning structural descriptions from a set of positive training instances. Specifically, we survey methods for finding the maximally-specific conjunctive generalizations (called MSC-generalizations) that characterize a given class of entities. This is one of the simplest learning problems that has been addressed by AI researchers. The problem of finding MSC-generalizations lends itself to comparative analysis because several different methods have been developed. This is unusual in current research on machine learning, which is currently investigating a wide variety of learning problems and learning methods. Particular methods reviewed in this chapter include those developed by Buchanan *et al.* [1971, 1976, 1978], , Hayes-Roth [1976a, 1976b, 1977, 1978] Vere [1975, 1977, 1978, 1980], Winston [1970, 1975], and the authors. This chapter is based on the article by Dietterich and Michalski [1981].

Before proceeding any further, let us explain our terminology. The chapter deals first of all with *structural descriptions*. Structural descriptions portray objects as composite structures consisting of various components. For instance, a structural description of a building could represent the building in terms of the floors, the walls, the ceilings, the hallways, the roof, and so forth, along with the relations that hold among these various components. Structural descriptions can be contrasted with *attribute descriptions*, which specify only global properties of an object. An attribute description of a building might list its cost, architect, height, total square-footage and so forth. No internal structure is represented. Attribute descriptions can be expressed using propositional logic—that is, null-ary predicates.[1] Structural descriptions, however, must be expressed in predicate logic. Each subcomponent is described globally using variables and unary predicates, and relations between components are expressed as k-ary predicates and functions.[2] In this chapter, variables, predicates, and functions are all referred to as *descriptors*.

The second item of terminology that requires explanation is the notion of a maximally-specific conjunctive generalization. A *conjunctive generalization* is a description of a class of objects obtained by forming the conjunction (AND) of a group of primitive statements. For example, the class of houses might be described as the set of all objects such that:

[1]This is a slight simplification. With multi-valued attributes such as color, one must either create a separate predicate for each color or else employ some form of multiple-valued logic, such as VL_1.

[2]This is also a slight simplification. In principle, it is always possible to convert a structural description into an attribute description, but such a conversion leads to a combinatorial explosion in the number of attributes.

the number of floors is less than four AND the purpose of the building is to be used as a dwelling

We write this symbolically as a VL_1 expression:

[#-of-floors < 4] & [purpose-of-building = dwelling]

An example of a description that is *not* conjunctive is the definition of "not married for tax purposes" as:

[marital status = single] $\vee$ [marital status = married] [filing status = separate returns]

This is a *disjunctive* description.

A *maximally-specific* conjunctive generalization is the most detailed (most specific) description that is true of all of the known objects in the class. Since specific descriptions list many facts about the class, the maximally-specific conjunctive generalization is the longest conjunctive generalization that still describes all of the training instances.

Now that we have described the scope of this chapter, we introduce several issues that are important in learning from examples. From these issues, we will later develop four criteria for evaluating learning systems and apply these criteria to the comparison of five existing learning methods.

3.1.2 Important Aspects of Learning From Examples

The process of inductive learning can be viewed as a search for plausible general descriptions (inductive assertions) that explain the given input data and are useful for predicting new data. In order for a computer program to formulate such descriptions, an appropriate description language must be used. For any set of input data and any non-trivial description language, a large number of inductive assertions can be formulated. These assertions form a set of descriptions partially ordered by the relation of relative generality [Mitchell, 1977]. The minimal elements of this set are the most specific descriptions of the input data in the given language, and the maximal elements are the most general descriptions of these data. The elements of this set can be generated by starting with the most specific descriptions and repeatedly applying rules of generalization to produce more general descriptions.

The view of induction as a search through a space of generalized descriptions draws attention to the following aspects of learning:

- **Representation.** What description language is employed for expressing the input examples and formulating the inductive assertions? What are the possible forms of assertions that a method is able to learn? What operators are used in these forms?
- **Type of description sought.** For what purpose are the inductive assertions being formulated? What assumptions does the induction method make about the underlying process(es) that generated the data?
- **Rules of generalization.** What kinds of transformations are performed on

the input data and intermediate descriptions in order to produce the inductive assertions?

- **Constructive induction.** Does the induction process change the description space; that is, does it produce new descriptors that were not present in the input events?
- **Control strategy.** What is the strategy used to search the description space: bottom-up (data-driven), top-down (model-driven), or mixed?
- **General versus problem-oriented approach.** Is the method oriented toward solving a general class of problems, or is it oriented toward problems in some specific application domain?

We now discuss each of these aspects in more detail.

3.1.3 Representation Issues

Many representational systems can be used to represent events and generalizations of events—for example, predicate calculus, production rules, hierarchical descriptions, semantic nets, frames, and scripts. Much AI work on inductive learning (the exceptions include the AM system [Lenat, 1976], and work by Winston [1970]) has employed predicate calculus (or some closely related system), because of its well-defined syntax and semantics. (An important study of theoretical problems of induction in the context of predicate calculus was undertaken by Plotkin [1970, 1971].)

The mere statement that some learning method "uses predicate calculus" does not tell us very much about that method. Most learning methods place further restrictions on the forms of inductive assertions. For example, although a learning system might in principle be able to represent disjunctive descriptions, in practice it may have no mechanisms for actually discovering such descriptions. One way to capture this distinction between "representable forms" and "learnable forms" is to indicate which operators can actually be used in each. The most common operators are conjunction (&), disjunction ($\vee$), exception, and the existential and universal quantifiers.

3.1.4 Types of Descriptions

Since induction is a search through a description space, one must specify the goal of this search—that is, one must provide criteria that define the goal description. These criteria depend upon the specific domain in question, but some regularities are evident. We distinguish among characteristic, discriminant, and taxonomic descriptions.

A *characteristic description* is a description of a class of objects (or situations, events, and so on) that states facts that are true of all objects in the class. It is usually intended to discriminate objects in the given class from objects in all other possible classes. For example, a characteristic description of the set of all tables would discriminate any table from all things that are non-tables. In this

way, the description *characterizes* the concept of a table. The task of discovering a characteristic description is a single-concept acquisition task (see Chapter 4 of this book). Since it is impossible to examine all objects in a given class (or *not* in a given class), a characteristic description is usually developed by specifying all characteristics that are true for all *known* objects of the class (positive examples). In some problems, negative examples (counterexamples) are available that represent objects known to be outside the class. Negative examples can greatly help to circumscribe the desired conceptual class. Even more helpful are counterexamples that are "near misses"—that is, negative examples that just barely fail to be positive examples (see Winston [1970, 1975]).

A *discriminant description* is a description of a class of objects in the context of a *fixed* set of other classes of objects. It states only those properties of the objects in the given class that are necessary to distinguish them from the objects in the other classes. A characteristic description can be viewed as an extreme kind of discriminant description in which the given class is discriminated against infinitely many alternative classes.

A *taxonomic description* is a description of a class of objects that subdivides the class into subclasses. In constructing such a description, it is assumed that the input data are not necessarily members of a single conceptual class. Rather it is assumed that they are members of several different classes (or produced by several different processes). An important kind of taxonomic description is a description that determines a *conceptual clustering*—a structuring of the data into object classes corresponding to distinct concepts. Taxonomic descriptions can be "flat"—with all object classes stated at the same level of abstraction—or hierarchical—with object classes arranged in an abstraction tree. A taxonomic description is fundamentally disjunctive. The overall class is described by the disjunction of the subclass descriptions. Taxonomic description is a kind of descriptive generalization rather than concept acquisition (see Chapter 4 of this book).

Determination of characteristic and discriminant descriptions is the subject of learning from (pre-classified) examples, while determination of taxonomic descriptions (conceptual clustering) is the subject of learning from observation or "learning without teacher". This distinction between these two forms of learning is examined in detail in Chapter 4 of this book.

In this chapter we restrict ourselves to the problem of determining characteristic descriptions. The problem of determining discriminant descriptions has been studied by Michalski and his collaborators [Larson & Michalski, 1977; Larson, 1977; Michalski, 1973, 1975, 1977, 1980a, 1980b] (see also Chapters 4 and 15 of this book.). A general method and computer program, CLUSTER/2, for conceptual clustering is described by Michalski and Stepp in Chapter 11 of this book.

3.1.5 Rules of Generalization

The partially-ordered space of descriptions of different levels of generality can be described by indicating what transformations are being applied to change less general descriptions into more general ones. Consequently, determination of inductive assertions can be viewed as a process of consecutive application of certain "generalization rules" to initial and intermediate descriptions. A generalization rule is a transformation rule that, when applied to a classification rule S_1 ::> K, produces a more general classification rule S_2 ::> K.[3] This means that the implication $S_1 \Rightarrow S_2$ holds. A generalization rule is called *selective* if S_2 involves no descriptors other than those used in S_1. If S_2 does contain new descriptors, then the rule is called *constructive* (see section 3.1.6). Selective rules of generalization do not change the space of possible inductive assertions, while constructive rules *do* change it.

The concept of rules of generalization provides further insight into the view of induction as a heuristic search of description space. The rules of generalization specify the operators that the search uses to move from one node to another in this space. The concept of generalization rules is also useful for comparing different learning methods because these rules abstract from the particular description languages used in the methods. In this chapter, we briefly outline the concept of a generalization rule and present a few examples. Chapter 4 presents a much more detailed discussion of the subject and an extensive list of generalization rules.

One of the simplest generalization rules is the *dropping condition* rule, which states that to generalize a conjunction, you may drop any of its conjunctive conditions. For example, the class K of "red apples" can be generalized to the class of all "apples" of any color by dropping the "red" condition. This can be written as:

red(*v*) & apple(*v*) ::> K can generalize to apple(*v*) ::> K

This is a selective rule of generalization because it does not introduce any new descriptors. An example of a constructive rule is the *find extrema of partial orders* rule. This rule augments a structural description by adding new descriptors for objects that are at the end points of ordered chains. For example, in a description of a four-storey office building, we might have the statement that "the second floor is on top of the first floor, the third floor is on top of the second, and so on." The find extrema rule would generate the fact that "the first floor is the bottom-most and the fourth floor is the top-most floor." The "on top of" relations form an ordered chain. Symbolically, this is written as:

ontop(f2,f1) & ontop(f3,f2) & ontop(f4,f3) |< most-ontop(f4) & least-ontop(f1)

[3]The notation S_1 ::> K means that all objects for which S_1 is true are classified as belonging to class K.

where the |< sign is interpreted as "can be generalized to". Other selective rules of generalization needed for this chapter include:

- the turning constants to variables rule
- the adding internal disjunction rule
- the closing interval rule
- the climbing generalization tree rule

These rules are explained in Chapter 4 of this book.

We also employ one rule of specialization. Any of the above rules of generalization can become rules of specialization by using them in reverse. However, one important rule of specialization is the introducing exception rule. It can be applied to a description in order to specialize it to take into account a counterexample. Suppose, for example, that a program is attempting to learn the concept of a "fish". Its initial hypothesis might be that a fish is anything that swims. However, it then is told about a dolphin that swims and breathes air but is not a fish. At this point, the program might guess that a fish is anything that swims and does not breathe air. This can be written as:

current description: swims(v) ::> K
negative example: swims(v) & breathes-air(v) ::> ~K
|> swims(v) & ~breathes-air(v) ::> K

The |> sign is interpreted as meaning "can be specialized to".

3.1.6 Constructive Induction

As we have mentioned above, constructive induction is any form of induction that generates new descriptors not present in the input data. It is important for learning programs to be able to perform constructive induction, since it is well known that many AI problems cannot be solved without a change of representation. Many existing methods of induction (for example, [Hunt *et al.*, 1966; Hayes-Roth, 1976a, 1976b; Vere, 1975, 1980; Mitchell, 1977, 1978]) do not perform constructive induction. We say that these methods perform *selective induction*, since the descriptors present in the generalizations produced by the program are selected from those present in the input data.

There are several existing systems that perform some form of constructive induction. Soloway's BASEBALL system [Soloway, 1978], for example, applies several rules of constructive induction to convert raw snapshots of a simulated baseball game into high-level episode descriptions that can be generalized to discover such concepts as "run", "hit", and "out". In this system, the constructive induction takes place first, followed by selective induction.

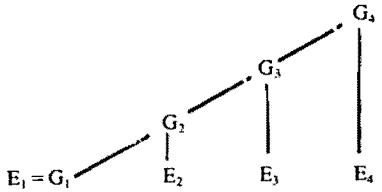
Larson's INDUCE-1 system [Larson, 1977; Larson & Michalski, 1977], on the other hand, performs constructive and selective induction simultaneously. INDUCE-1 implements the "find extrema of partial orders" rule of generalization described above, along with a few other constructive induction rules. New

descriptors are tested for discriminatory ability before they are added to all of the training instances.

Unfortunately, most existing systems have not implemented constructive induction rules in any general way. Instead, specific procedures are written to generate the new descriptors. This is an important problem for future research. In Chapter 4 of this book, Michalski presents more rules of constructive induction.

3.1.7 Control Strategy

Induction methods can be divided into bottom-up (data-driven), top-down (model-driven), and mixed methods depending on the strategy that they employ during the search for generalized descriptions. Bottom-up methods process the input events one at a time, gradually generalizing the current set of descriptions until a final conjunctive generalization is computed:



G_2 is the set of conjunctive generalizations of E_1 and E_2. G_1 is the set of conjunctive generalizations obtained by taking each element of G_{1-1} and generalizing it with E_1.

Methods described by Winston, Hayes-Roth, and Vere are reviewed in this chapter. Other bottom-up methods include the candidate elimination approach described by Mitchell [1977, 1978], the ID3 technique of Quinlan [1979a, 1979b] (see also Chapter 15 of this book), and the Uniclass method described by Stepp [1970].

Top-down methods search a set of possible generalizations in an attempt to find a few "best" hypotheses that satisfy certain requirements. The two methods discussed in this chapter (Buchanan, *et al.* and Michalski) search for a small number of conjunctions that together cover all of the input events. The search proceeds by choosing as the initial working hypotheses some elements from the partially-ordered set of all possible descriptions. If the working hypotheses satisfy certain criteria, then the search halts. Otherwise, the current hypotheses are modified by slightly generalizing or specializing them. These new hypotheses are then checked to see if they satisfy the termination criteria. The process of modifying and checking continues until the criteria are met. Top-down techniques typically have better noise immunity and can be easily extended to discover disjunctions. The principal disadvantage of these techniques is that the working hypotheses must be checked repeatedly to determine whether they subsume all of the input events.

3.1.8 General versus Problem-oriented Methods

It is a common view that general methods of formal induction, although mathematically elegant and theoretically applicable to many problems, are in practice very inefficient and rarely lead to any interesting solutions. This opinion has led certain workers to abandon (at least temporarily) work on general methods and concentrate on learning problems in some specific domains (for example, Buchanan, *et al.* [1978] in chemistry or Lenat [1976] in elementary number theory). Such an approach can produce novel and practical solutions. On the other hand, it is difficult to extract general principles of induction from such problem-specific work. It is also difficult to apply such special-purpose programs to new areas.

An attractive possibility for solving this dilemma is to develop methods that incorporate various general principles of induction (including constructive induction) together with mechanisms for using exchangeable packages of problem-specific knowledge. This idea underlies the development of the INDUCE programs [Larson, 1977; Larson & Michalski, 1977; Michalski, 1980a] and the Star methodology described by Michalski in Chapter 4 of this book.

3.2 COMPARATIVE REVIEW OF SELECTED METHODS

3.2.1 Evaluation Criteria

The selected methods of induction are evaluated in terms of several criteria considered especially important in view of our discussion in Section 3.1.

1. *Adequacy of the representation language:* The language used to represent input data and output generalizations determines to a large extent the quality and utility of the output descriptions. Although it is difficult to assess the adequacy of a representation language out of the context of some specific problem, recent work in AI has shown that languages that treat all phenomena uniformly must sacrifice descriptive precision. For example, researchers who are attempting to build systems for understanding natural language prefer rich knowledge representations, such as frames, scripts, and semantic nets, to more uniform and less structured representations, such as attribute-value lists and PLANNER-style representations. Although languages with many syntactic forms do provide greater descriptive precision, they also lead to combinatorial increases in the complexity of the induction process. In order to control this complexity, a compromise must be sought between uniformity and richness of representational forms. In the evaluation of each method, a review of the operators and syntactic forms of each description language is provided.

2. *Rules of generalization implemented:* The generalization rules implemented in each algorithm are listed.

3. *Computational efficiency:* To get some approximate measure of computational

efficiency, we have hand simulated each algorithm on the test problem shown in Figure 3-2. In the simulation, we have measured the total number of times an inductive description was generated and the total number of times one inductive description was compared to another (or compared to a training instance). These provide good measures of computational effort, since generation and comparison of structural descriptions are expensive operations. We have also computed the ratio of the number of final descriptions output by the algorithm to the total number of descriptions generated by the algorithm. This provides a measure of overall efficiency, since a ratio of 1 indicates that every description generated by the algorithm was correct, while a ratio of 0 indicates that none of the generated descriptions were correct.

Our evaluation of these induction methods is not based entirely on these numerical measures, however (particularly since they are derived from only one test problem). An additional value of the simulation is that it gives some general idea of how the algorithms behave and shows the kinds of descriptions that the algorithms are able to discover. The reader is admonished to treat the efficiency measurements as highly approximate.

4. *Flexibility and extensibility:* Programs that can only discover conjunctive characteristic descriptions have limited practical application. In particular, they are inadequate in situations involving noisy data or in which no single conjunctive description can describe the phenomena of interest. Consequently, as one of the evaluation criteria, we consider the ease with which each method could be extended to:

- discover descriptions with forms other than conjunctive generalizations, for example, disjunctions and exceptions (see Section 3.1.4)
- include mechanisms that facilitate the detection of errors in the input data
- provide a general facility for incorporating externally-specified domain knowledge into the induction process as an exchangeable package
- perform constructive induction

Two sample learning problems will be used to explain these methods. The first problem (Figure 3-1) is made up of two examples (E1 and E2). Each example consists of objects (geometrical figures) that can be described by:

- attributes *size* (small or large) and *shape* (circle or square)
- relationships *ontop* (which indicates that one object is above another) and *inside* (which indicates that one object lies inside another)

The second sample problem (Figure 3-2) contains three examples of constructions made of simple geometrical objects. These objects can be described by:

- attributes *shape* (box, triangle, rectangle, ellipse, circle, square, or diamond), *size* (small, medium, or large), and *texture* (blank or shaded)
- relationships *ontop* and *inside* (the same as in the first sample problem)

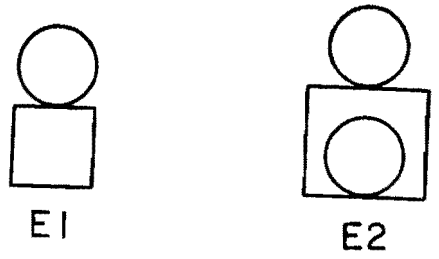


Figure 3-1: Sample problem for illustrating representation languages.

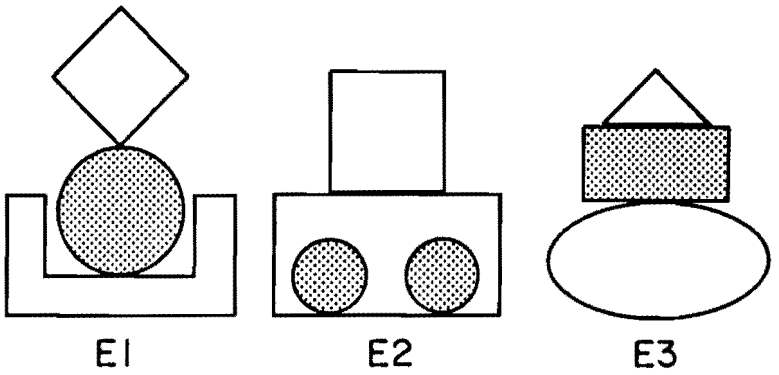


Figure 3-2: Sample problem for comparing the performance of the methods.

In each sample problem, the task is to determine a set of maximally-specific conjunctive generalizations (MSC-generalizations) of the examples. No negative examples are supplied in either problem. In the discussion below, the first problem is used to illustrate the representational formalism and the generalization process implemented in each method. The second, more complex, problem is used to compare the computational efficiency and representational adequacy of each method. This comparison is based on a hand simulation of each method.

3.2.2 Data-driven Methods: Winston, Hayes-Roth, and Vere

3.2.2.1 Winston: Learning Blocks World Concepts

Winston's well known work [Winston, 1970, 1975] deals with learning concepts that characterize simple toy block constructions. Although his method uses no precise criterion to define the goal description, the method usually develops MSC-generalizations of the input examples. The method assumes that the examples are provided to the program by an intelligent teacher who carefully chooses both the kinds of examples used and their order of presentation. The

program uses so-called "near miss" negative examples to rapidly determine the correct generalized description of the concept. A near-miss example is a negative example that differs from the desired concept in only one significant attribute. Winston also uses the near-misses to develop "emphatic" conditions such as "must support" or "must not support". These *Must-* type descriptors indicate which conditions in the concept description are necessary to eliminate negative examples.

As Knapman has pointed out in his review of Winston's work [Knapman, 1978], many parts of the exposition in Winston's thesis [Winston, 1970] and subsequent publication [Winston, 1975] are not entirely clear. Although the general ideas in the thesis are well-explained, the exact implementation of these ideas is difficult to extract from these publications. Consequently, our description of Winston's method is necessarily a reconstruction. We begin by discussing the knowledge representation employed by Winston. Then, we turn our attention to his learning algorithm.

A semantic network is used to represent the input events, the background blocks-world knowledge, and the concept descriptions generated by the program (see Figures 3-3 and 3-4). The representation is quite general although the implemented programs appear to process the network in domain-specific ways (see Knapman [1978]; Winston [1970, page 196]).

Nodes in the network are used for several different purposes. We will illustrate these purposes by referring to the corresponding concepts in first-order predicate logic (FOPL). The first use of nodes is to represent various primitive concepts that are properties of objects or their parts (such as *small*, *size*, *circle*, *shape*). Nodes in this case correspond to constants in first-order predicate logic expressions. There is no distinction between attributes and values of attributes in Winston's network representation, and consequently, there is no representational equivalent of the one-argument predicates and functions of FOPL.

Another use of nodes is to represent individual examples and their parts. Thus, in Figure 3-3, we have the node E1 and two nodes A and B that make up E1. These can be regarded as quantified variables in predicate calculus. Distinct variable nodes are created for each training example.

Labeled links connecting these nodes represent various binary relationships among the nodes. The links correspond to two-argument predicates. The first two uses of nodes as constants and variables, plus the standard use of links as predicates, constitute the basic semantic network representation used by Winston.

There is, however, a third use of nodes. Each link type (analogous to a predicate symbol) is also represented in the network as a node. Thus, in addition to the numerous *On-Top* links that may appear in the network, there is an *On-Top* node that describes the link type *On-Top* and its relationship to other link types. For example, there might be a *Negative-Satellite* link that joins the *On-Top* node to the *Beneath* node. Such a link indicates that *On-Top* and *Beneath* are semantically opposite predicates. Similarly, there is a Must-be-Satellite link connecting the *Must-Be-On-Top* node to the *On-Top* node.

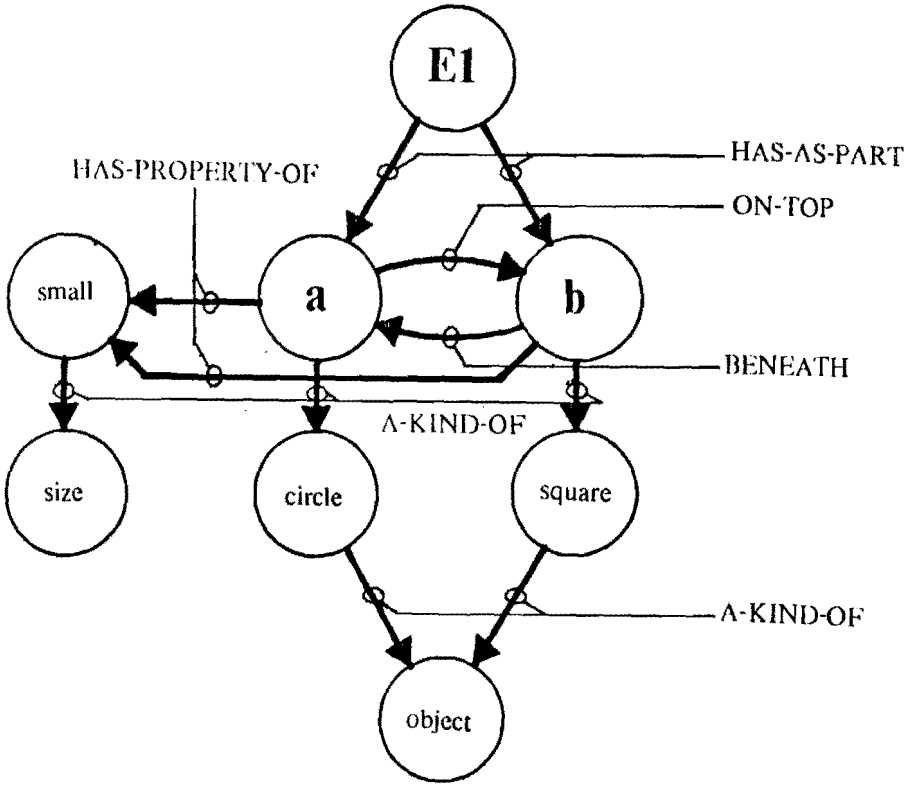


Figure 3-3: Network representing example E1 in Figure 3-1.

All of the nodes in the network are joined into one generalization hierarchy through the *A-Kind-Of* links. This hierarchy is used to implement the climbing generalization tree rule.

Now that we have described the network representation, we turn our attention to the learning algorithm. The learning algorithm proceeds in two steps. First, the current concept description is compared to the next example, and a difference description is developed. Then this difference description is processed to obtain a new, generalized concept description. Often, the second step results in several possible generalized concept descriptions. In such a case, one generalized concept is selected for further refinement and the remaining possibilities are placed on a backtrack list. The program backtracks when it is unable to consistently generalize its current concept description.

The first step of the algorithm (the development of the difference

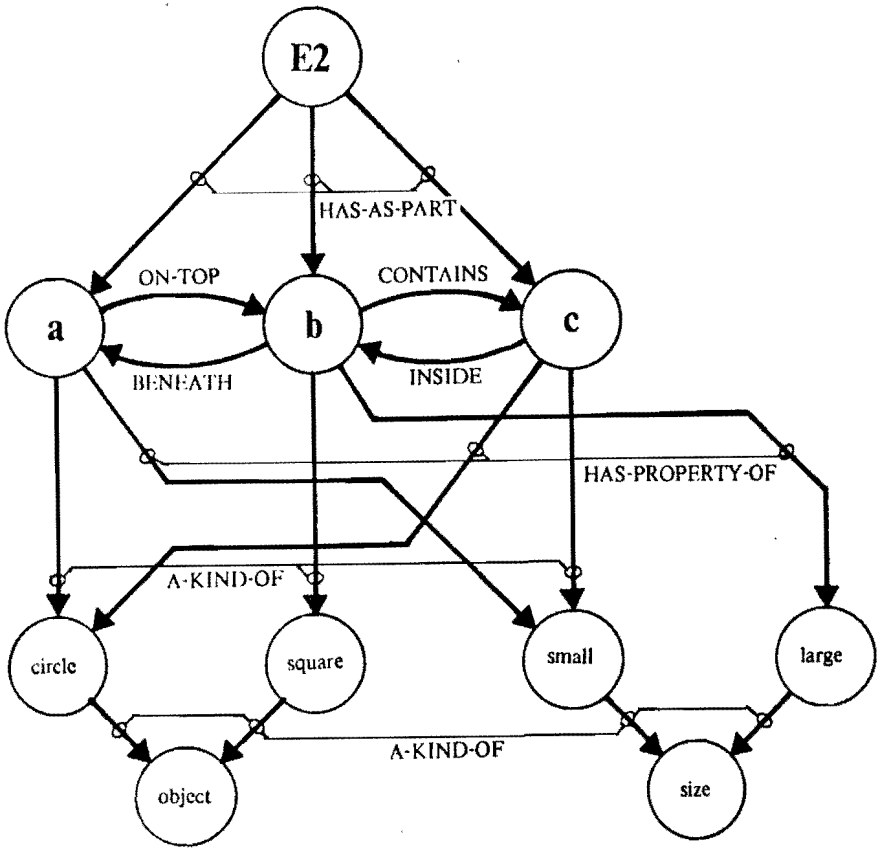


Figure 3-4: Network representing example E2 in Figure 3-1.

description) is accomplished by graph-matching the current concept description against the example supplied by the teacher, and annotating this match with comment notes (C-NOTES). These C-NOTES describe conditions in the concept description and example that partially matched or did not match. Winston's description of the graph-matching algorithm is sketchy [Knapman, 1978; Winston, 1970, pages 254-263]. The algorithm apparently finds one "best" match between the training example and the current concept description. The method does not address the important problem of multiple graph sub-isomorphisms, that is, the problem arising when the training example matches the current concept description in more than one way. This problem was apparently avoided by assuming that the teacher will present training instances that can be unambiguously matched to the current concept description.

Once this match between the concept description and the example is obtained, a generalized skeleton is created containing only those links and nodes that matched exactly. The C-NOTES are then attached to this skeleton. Each C-NOTE is a sub-network of nodes and links that describes a particular type of match. There are several types of C-NOTES corresponding to partially-matching or mismatching nodes and partially-matching or mismatching links. The different types are summarized in Table 3-1. In detail, there are the following types of C-NOTES:

- For nodes:
  - *Intersection* C-NOTES indicate that two nodes match exactly.
  - *A-Kind-of-Merge* and *A-Kind-Of-Chain* C-NOTES indicate that two nodes match partially. The *A-Kind-Of-Merge* C-NOTE handles the case when two nodes are different but share a common *A-Kind-Of* link, for example, when *square* partially matches *triangle* (since they are both polygons). The *A-Kind-Of-Chain* C-NOTE handles the case when a node matches a more general node, for example, when *square* matches *polygon*.
  - *Exit* C-NOTES indicate that two nodes do not match at all.
- For links:
  - *Negative-Satellite-Pair* C-NOTES indicate that two semantically opposite links mismatched, for example, *Marries* and *Does-Not-Marry*.
  - *Must-Be-Satellite-Pair* C-NOTES indicate that a normal link, such as *Supports*, matches an emphatic link, such as *Must-Support*.
  - *Must-Not-Be-Satellite-Pair* C-NOTES indicate that a normal link matches a *Must-Not* form of the same link.
  - *Supplementary Pointer* C-NOTES indicate that two links do not match at all.

Table 3-1: Winston's C-NOTE Categories

	Match	Partially match	Mismatch
Node	Intersection	A-Kind-Of-Merge A-Kind-Of Chain	Exit
Link		Negative-Satellite-Pair Must-Not-Be-Satellite-pair	Supplementary pointer

The network diagram of Figure 3-5 shows the difference description that results from matching the two networks of Figures 3-3 and 3-4 to each other.

The generalization phase of the algorithm is fairly simple. Each C-NOTE is handled in a way determined by the C-NOTE type and whether the example is a positive or negative training example. Winston provides a table that indicates what actions his program takes in each case [Winston, 1970, pages 145-146].

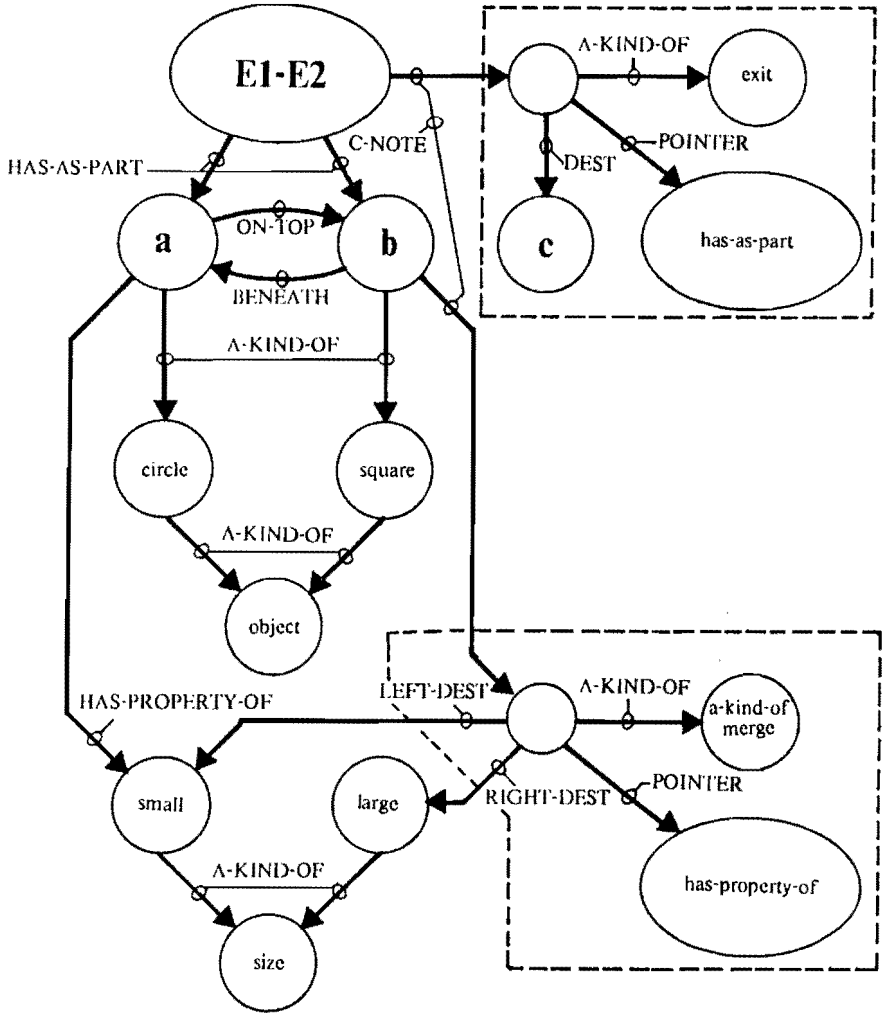


Figure 3-5: Difference description obtained by comparing E1 and E2 from Figure 3-1 and annotating the comparison with two C-NOTES.

Some C-NOTES can be handled in multiple ways. For positive examples, only one C-NOTE causes problems: the *A-Kind-Of-Merge*. In this case, the program can either climb the *A-Kind-Of* generalization tree or else drop the condition altogether. The program develops both possibilities but only pursues the former (leaving the latter on the backtrack list). The concept description that results from generalizing the difference description of Figure 3-5 is shown in

Figure 3-6. The alternative generalization would drop the *Has-Property* link from node *b*.

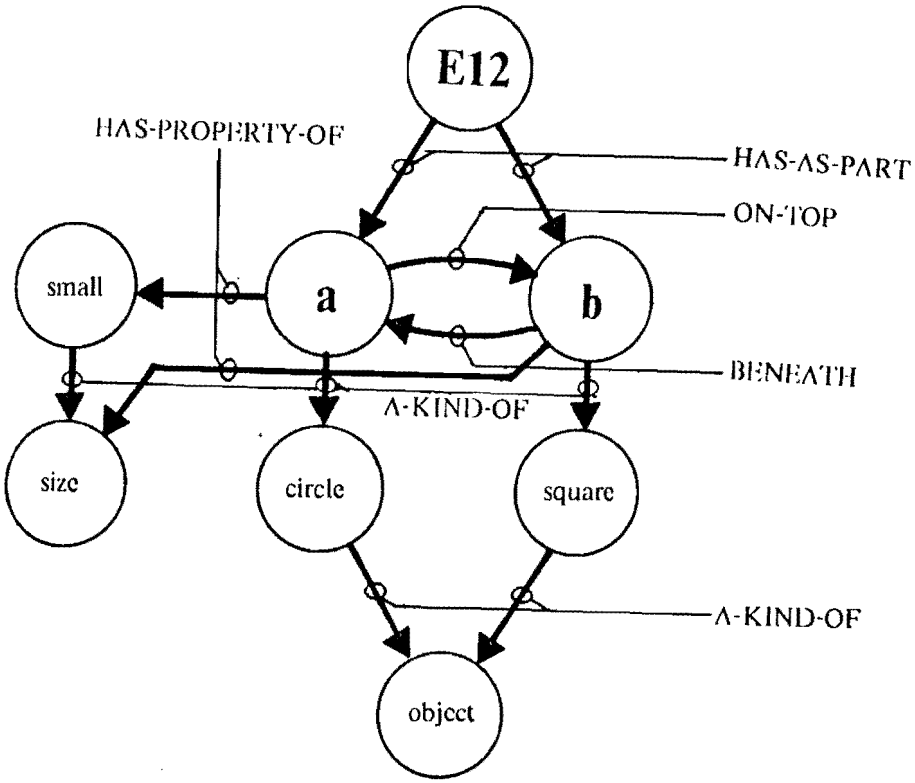


Figure 3-6: Network representing the generalized concept resulting from generalizing the difference description of Figure 3-5.

Evaluation:

1. *Representational adequacy.* The semantic network is used to represent properties, object hierarchies (using *A-Kind-Of*), and binary relationships. As in most semantic networks, n-ary relationships cannot be represented directly. The conjunction operator is implicit in the structure of the network, since all of the conditions represented in the network are assumed to hold simultaneously. There is no mechanism indicated for representing disjunction or internal disjunction. The *Not* and *Must-Not* links implement a form of the exception operator. An interesting feature of Winston's work is the use of the emphatic *Must-* relationships.

The program works in a depth-first fashion and produces only one generalized concept description for any given order of the training examples. Permuting the training examples may lead to a different generalization. Two generalizations obtained by simulating Winston's learning algorithm on the examples of Figure 3-2 are shown in Figures 3-7 and 3-8.

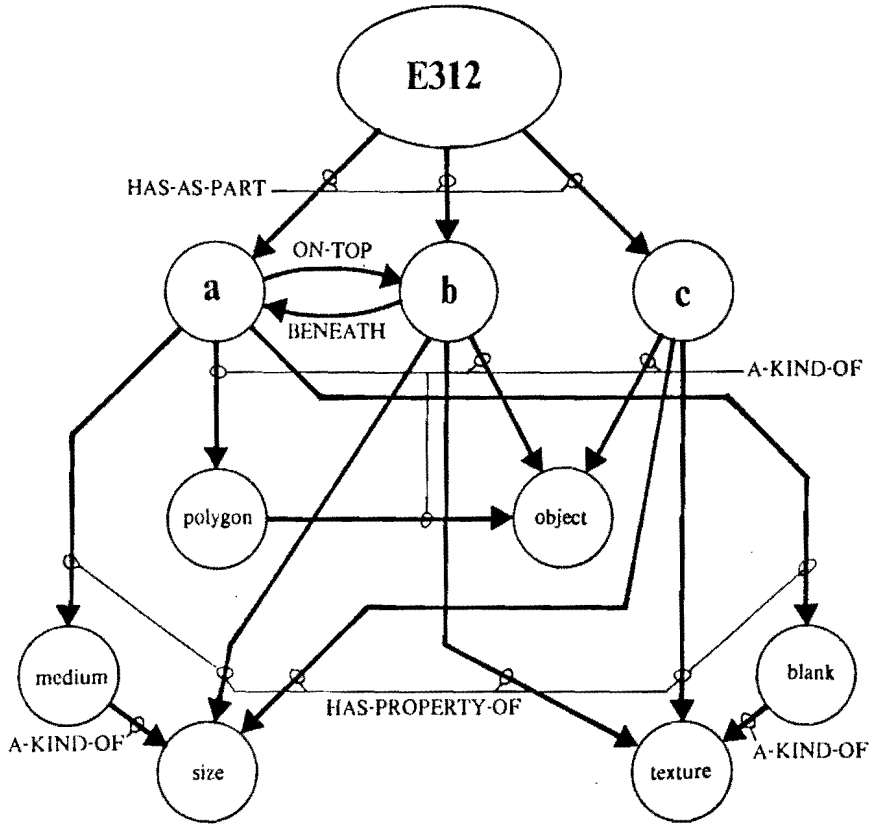


Figure 3-7: The first generalization obtained by simulating Winston's learning algorithm on the examples of Figure 3-2 (in the order E3, E1, E2). An English paraphrase is: "There is a medium, blank polygon on top of another object that has a size and texture. There is also another object with size and texture."

The second generalization (Figure 3-8) is not maximally specific since it does not mention the fact that all training examples also contain a small- or medium-sized shaded object. The algorithm cannot discover this generalization

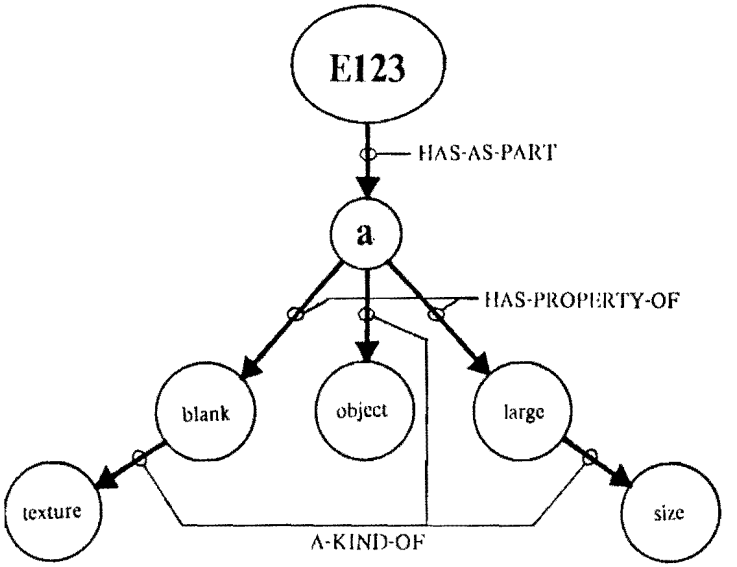


Figure 3-8: The second generalization obtained by simulating Winston's learning algorithm on the examples of Figure 3-2 (in the order E1, E2, E3). An English paraphrase is: "There is a large, blank object."

due to the fact that the graph-matcher finds the "best" match of the current concept with the example. When the order of presentation of the examples is E1 followed by E2 followed by E3, the "best" match of the first two examples eliminates the possibility of discovering the maximally-specific conjunctive generalization when the third example is matched.

2. *Rules of Generalization.* The program uses the dropping condition rule (for generalizing exit C-NOTES), the turning constants to variables rule (when creating the generalized skeleton), and the climbing generalization tree rule (for the *A-Kind-Of-Merge*). It also uses the introducing exception specialization rule (for the *A-Kind-Of-Merge* C-NOTE with negative examples).

3. *Computational efficiency.* The algorithm is quite fast: it requires only two graph comparisons to handle the examples of Figure 3-2. However, the algorithm does use a lot of memory to store intermediate descriptions. The first graph comparison produces eight alternatives, of which only one is pursued. The second graph comparison leads to four more alternatives from which one is selected as the "best" concept description. This inefficient use of memory is reflected in our figure for computational efficiency (the number of output descriptions / the number of examined descriptions), which is 1/11 or 9%.

The performance of the algorithm can be much worse in certain situations. When "poor" negative examples are used—those which do not match the current concept description well—the number of intermediate descriptions explodes combinatorially. Such situations are also likely to cause extensive backtracking.

Since the algorithm produces only one generalization for any given order of the input examples, it must be executed repeatedly if several alternative generalizations are desired.

4. *Flexibility and Extensibility.* Iba [1979] has successfully extended this algorithm to discover some disjunctive descriptions. His solution is not entirely general, however. The main difficulty seems to be that Winston's algorithm operates under the assumption that there is one conjunctive concept characterizing the examples, so the development of disjunctive concepts is not consistent with the spirit of the work.

Since the program behaves in a depth-first manner, noisy training events cause it to make serious errors from which it cannot recover without extensive backtracking. This is not surprising since Winston assumes that the teacher is intelligent and does not make any mistakes in training the student. It seems to be very difficult to extend this method to handle noisy input data.

The inductive generalization portion of the program does not contain much problem-specific knowledge. However, many of the techniques used in the program, such as building complete difference descriptions and using a backtracking search, may become combinatorially infeasible in real-world problem domains. The *A-Kind-Of* generalization hierarchy can be used to represent problem-specific knowledge.

The system of programs described by Winston performs some types of constructive induction. The original inputs to the system are noise-free line drawings. Some knowledge-based algorithms convert these line drawings into the network representation. Winston describes an algorithm for combining a group of objects into a single concept and subsequently using this concept in other descriptions. The "arcade" concept ([Winston, 1970], page 183) is a good example of such a constructive induction process.

3.2.2.2 Hayes-Roth: Program SPROUTER

Hayes-Roth's work on inductive learning [Hayes-Roth, 1976a, 1976b; Hayes-Roth & McDermott, 1977, 1978] is concerned with finding MSC-generalizations of a set of input positive examples (he calls such generalizations *maximal abstractions* or *interference matches*). *Parameterized structural representations* (PSR's) are used to represent both the input events and their generalizations. The PSR's for the two events of Figure 3-1 are:

E1: {{circle:a}{square:b}{small:a}
{small:b}{ontop:a, under:b}}

E2: {{circle:c}{square:d}{circle:e}
{small:c}{large:d}{small:e}
{ontop:c, under:d}{inside:e, outside:d}}

E1: {{circle:a}{square:b}{small:a}
{small:b}{ontop:a, under:b}}

E2: {{circle:c}{square:d}{circle:e}
{small:c}{large:d}{small:e}
{ontop:c, under:d}{inside:e, outside:d}}

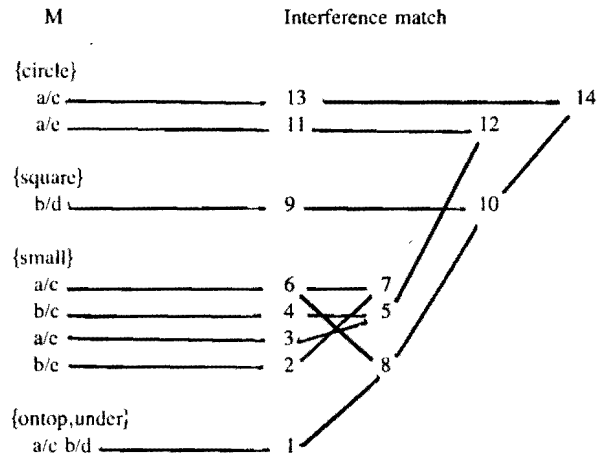
In Hayes-Roth's terminology, the expressions such as {small:a} are called case frames. They are composed of case labels (such as small, circle) and parameters (such as a, b, c, d). The PSR can be interpreted as a conjunction of predicates of the form case-label(parameter-list). For example, {small:a} can be interpreted as small(a), and {ontop:c, under:d} can be interpreted as ontop(c,d). The parameters can be viewed as existentially-quantified variables denoting distinct objects.

The induction algorithm works in a purely bottom-up fashion. The first set of conjunctive generalizations, G_1, is initialized to contain only the first input example. Given a new example and the set of generalizations, G_i, obtained in the i^{th} step, a new set of generalizations, G_{i+1}, is obtained by performing a partial match between each element in G_i and the current training example. It is not clear from publications [Hayes-Roth, 1976b; Hayes-Roth, 1976a; Hayes-Roth & McDermott, 1977; Hayes-Roth & McDermott, 1978] whether or not these sets G_i are pruned during this process. Hayes-Roth calls each of the partial-matching operations an *interference match.*

The interference match attempts to find the longest one-to-one match of parameters and case frames (that is, the longest common subexpression). This is accomplished in two steps. First the case frames in E1 and E2 are matched in all possible ways to obtain the set M. Two case frames match if all of their case labels match. Each element of M is a case frame and a list of parameter correspondences that permit that case frame to match in both events:

M = {{circle:((a/c)(a/e))},
{square:((b/d))},
{small:((a/c)(b/c)(a/e)(b/e))},
{ontop,under:((a/c b/d))}}

The second step involves selecting a subset of the parameter correspondences in M such that all parameters can be bound consistently. This is conducted by a breadth-first search of the space of possible bindings with pruning of unpromising nodes. The search can be visualized as a node-building process. Here is one such (pruned) search graph:



The nodes are numbered in order of their generation. One at a time, a pair of corresponding parameters is selected from M and a new node is created for them. Then this new node is compared with all previously generated nodes. Additional nodes are created for each case in which the new parameter correspondence node can be consistently merged with a previously existing node. In the search graph above, when the parameter binding {small: (a/c)} is selected, node 6 is created. Then node 6 is compared to nodes 1 through 5 and two new nodes are created: node 7, which is created by merging node 6 (a/c) with node 2 (b/e), and node 8, which is created by merging node 6 (a/c) with node 1 (a/c b/d). Node 6 cannot be merged with node 3, for instance, because parameter a would be inconsistently bound to both parameters c and e.

When the search is completed, nodes 7, 12, and 14 are bindings that lead to conjunctive generalizations. Node 14, for example, binds a to c (to give v1) and b to d (to give v2) to produce the conjunction:

{{circle:v1}{square:v2}{small:v1}{ontop:v1, under:v2}}

The node-building process is guided by computing a utility value for each candidate node to be built. The nodes are pruned by setting an upper limit on the total number of possible nodes and pruning nodes of low utility when that limit is reached.

Evaluation:

1. *Representational adequacy*. The algorithm discovers the following conjunctive generalizations of the example in Figure 3-2:

a. {{ontop:v1, under:v2}{medium:v1}{blank:v1}}
There is a medium blank object ontop of something.

b. {{ontop:v1, under:v2}{medium:v1}{large:v2}{blank:v2}}
There is a medium object ontop of a large, blank object.

c. {{medium:v1}{blank:v1}{large:v3}{blank:v3}{shaded:v2}}

There is a medium sized blank object, a large sized blank object, and a shaded object.

PSR's provide two symbolic forms: parameters and case labels. The case labels can express ordinary predicates and relations easily. Symmetric relations may be expressed by using the same label twice as in {same!size:a, same!size:b}. The only operator is the conjunction. The language has no disjunction or internal disjunction. As a result, the fact that each event in Figure 3-2 contains a polygon on top of a *circle or rectangle* cannot be discovered.

2. *Rules of generalization*. The method uses the dropping condition and turning constants to variables rules.

3. *Computational efficiency*. On our test example, the algorithm requires 22 expression comparisons and generates 20 candidate conjunctive generalizations of which 6 are retained. This gives a figure of 6/20 or 30% for computational efficiency. Four separate interference matches are required since the first match of E1 and E2 produces three possible conjunctive generalizations.

4. *Flexibility and extensibility*. An attempt has been made (Hayes-Roth, personal communication) to extend this method to produce disjunctive generalizations and to detect errors in data. Hayes-Roth has applied this method to various problems in the design of the speech understanding system HEARSAY II. However, no facility has been developed for incorporating domain-specific knowledge into the generalization process.

Also, no facility for constructive induction has been incorporated although Hayes-Roth has developed a technique for converting a PSR to a lower-level, finer-grained uniform PSR. This transformation permits the program to develop descriptions that involve a many-to-one binding of parameters.

3.2.2.3 Vere: Program Thoth

Vere's earlier work on inductive learning [Vere, 1975] was also directed at finding the MSC-generalizations of a set of input positive examples (in his work such generalizations are called *maximal conjunctive generalizations* or *maximal unifying generalizations*). Each example is represented as a conjunction of *literals*. A *literal* is a list of constants called *terms* enclosed in parentheses. For example, the objects in Figure 3-1 would be described as:

E1: (circle a)(square b)(small a)(small b)(ontop a b)

E2: (circle c)(square d)(circle e)(small c) (large d)(small e)(ontop c d)(inside e d)

Although these resemble Hayes-Roth's PSR's, they are quite different. There are no distinguished symbols. All terms (such as "small" and "e") are treated uniformly.

As in Hayes-Roth's work, Vere's method operates in a purely bottom-up fashion in which the input examples are processed one at a time in order to build the set of conjunctive generalizations. The algorithm for generalizing a pair of events operates in four steps. First, the literals in each of the two events are

matched in all possible ways to generate the set of matching pairs MP. Two literals match if they contain the same number of constants and they share at least one common term in the same position. For the sample problem of Figure 3-2, we have:

```
MP = {((circle a),(circle c)),
      ((circle a),(circle e)),
      ((square b),(square d)),
      ((small a),(small c)),
      ((small a),(small e)),
      ((small b),(small c)),
      ((small b),(small e)),
      ((ontop a b),(ontop c d))}
```

The second step involves selecting all possible subsets of MP such that no single literal of one event is paired with more than one literal in another event. Each of these subsets eventually forms a new generalization of the original events.

In the third step, each subset of matching pairs selected in step 2 is extended by adding to the subset additional pairs of literals that did not previously match. A new pair p is added to a subset S of MP if each literal in p is related to some other pair q in S by a common constant in a common position. For example, if S contained the pair ((square b),(square d)) then we could add to S the pair ((ontop a b),(inside e d)) because the third element of (ontop a b) is the second element of (square b) and the third element of (inside e d) is the second element of (square d) (Vere calls this a 3-2 relationship). New indirectly-related pairs are merged into S until no more can be added.

In the fourth, and final, step, the resulting set of pairs is converted into a new conjunction of literals by merging each pair to form a single literal. Terms that do not match are turned into new terms, which may be viewed formally as variables. For example, ((circle a),(circle c)) would be converted to (circle v1).

Evaluation:

1. *Representational adequacy.* When applied to the test example (Figure 3-2) this algorithm produces many generalizations. A few of the significant ones are listed below:

- (ontop v1 v2)(medium v1)(large v2)(blank v2)(blank v3)(shaded v4) (v5 v4)

  There is a medium object on top of a large blank object. Another object is blank. There is a shaded object. (The literal (v5 v4) is *vacuous* since it contains only variables. Variable v5 was derived by unifying circle and triangle).

- (ontop v1 v2)(blank v1)(medium v1)(v9 v1)(v5 v3 v4)(shaded v3) (v7 v3)(v6 v3)(blank v4)(large v4)(v8 v4)

  There is a medium, blank object on top of some other object and there are two objects related in some way (v5) such that one is shaded and the other is large and blank.

- (ontop v1 v2)(medium v1)(blank v2)(large v2)(v5 v2)(shaded v3)(v7 v3) (blank v4)(v6 v4)

  There is a medium object on top of a large blank object. There is a shaded object and there is a blank object.

The representation is basically an uninterpreted list structure and, consequently, has very little logical structure. By convention the first symbol of a literal can be interpreted as a predicate symbol. The algorithm, however, treats all terms uniformly. This absence of semantic constraints creates difficulties. One difficulty is that the algorithm generates vacuous literals in certain situations. For instance, step 3 of the algorithm allows (circle a) to be paired with (triangle b) to produce the vacuous literal (v5 v4) as in generalization 1 above. Although these vacuous literals could easily be removed after being generated, the algorithm would perform more efficiently if it did not generate them in the first place. A second difficulty resulting from the relaxation of semantic constraints is that the algorithm creates generalizations involving a many-to-one binding of variables. While such generalizations may be desirable in some situations, they are usually meaningless, and their uncontrolled generation is computationally expensive.

The description language contains only the conjunction operator. No disjunction or internal disjunction is included.

2. *Rules of generalization.* The algorithm implements the dropping condition rule and the turning constants to variables rule.

3. *Computational efficiency.* From the published articles [Vere, 1975, 1977, 1978, 1980] it is not clear how to perform steps 2 and 3. The space of possible subsets of MP (computed in step 2) is very large, and the space of possible extensions to that set (computed in step 3) is even larger. An exhaustive search could not possibly give the computation times that Vere has published.

4. *Flexibility and extensibility.* Vere has published algorithms that discover descriptions with disjunctions [Vere, 1978] and exceptions (which he calls counterfactuals, see [Vere, 1980]). He has also developed techniques to generalize relational production rules [Vere, 1977, 1978]. The method has been demonstrated using the traditional AI toy problems of IQ analogy tests and blocks-world sequences. A facility for using background information to assist the induction process has also been developed. It uses a spreading activation technique to extract relevant relations from a knowledge base and add them to the input examples prior to generalizing them. The method has been extended to discover disjunctions and exceptions. It is not clear how well the method would work in noisy environments.

3.2.3 Model-driven Methods: Buchanan, *et al.*, and Michalski

In addition to acquiring context-free concept descriptions, some systems use models of the underlying domain to constrain the search for viable structural descriptions.

3.2.3.1 Buchanan, *et al.*: Program META-DENDRAL

META-DENDRAL is a program that discovers cleavage rules to explain the operation of a mass spectrometer. A mass spectrometer is a device that bombards small chemical samples with accelerated electrons, causing the molecules of the sample to break apart into many charged fragments. The masses of these fragments can then be measured to produce a *mass spectrum*—a histogram of the number of fragments (also called the *intensity*) plotted against their mass-to-charge ratio.

Analytic chemists can use the mass spectrum to guess the three-dimensional structure of the molecules in the sample. An expert system has been developed—the Heuristic DENDRAL program—that can also perform this structure elucidation task. It is supplied with the chemical formula (but not the three-dimensional structure) of the sample and its mass spectrum. Heuristic DENDRAL first examines the spectrum to obtain a set of constraints. These constraints are then given to CONGEN, a program that can generate all possible chemical structures satisfying the constraints. Finally, each of these generated structures is tested by running it through a mass-spectrometer simulator. The simulator applies a set of *cleavage rules* to predict which bonds in the proposed structure will be broken. The result is a simulated mass spectrum for each candidate structure. The simulated spectra are compared with the actual spectrum, and the structure whose simulated spectrum best matches the actual spectrum is ranked as the most likely structure for the unknown sample. The purpose of the META-DENDRAL system is to learn cleavage rules for use by the mass-spectrometer simulator.

The cleavage rules employed by the simulator are written as condition-action rules in which the condition part describes—in common ball-and-stick language—a portion of the molecular structure, and the action part indicates (by **) one or more bonds that will break (see Figure 3-9). The simulator applies these rules by matching the condition part against the molecular structure of the molecule being bombarded. Whenever the condition part matches, the simulator predicts that the bonds corresponding to those mentioned in the action part will break.

Figure 3-9 shows a typical cleavage rule. The atom descriptors have the following meanings. *Type* is the atomic element of the atom. *Nhs* is the number of hydrogen atoms bound to that atom. *Nbrs* is the number of non-hydrogen atoms bound to the atom. *Dots* counts the number of unsaturated valence electrons of the atom. This rule says that whenever a molecule containing the four atoms w, x, y, and z (connected as shown in the molecule graph and with the indicated atom descriptors) is placed in a mass spectrometer, then the bond joining w to x will be broken.

How can META-DENDRAL discover these rules? META-DENDRAL is given as input a set of molecules whose three-dimensional structures and mass spectra are known. We can view these training instances as condition-action rules of the form:

CONDITION PART (BOND ENVIRONMENT):

Molecule graph: w—x—y—z—

Atom descriptors:

atom	type	nhs	nbrs	dots
w	carbon	3	1	0
x	carbon	2	2	0
y	nitrogen	1	2	0
z	carbon	2	2	0

ACTION PART (CLEAVAGE PREDICTION):

w ** x—y—z—

Figure 3-9: Typical Cleavage Rule.

<whole molecular structure> ⇒ <mass spectrum>

The first step in META-DENDRAL (carried out by subprogram INTSUM) is to apply background knowledge and rules of constructive induction to convert these training instances into the form of highly-specific cleavage rules:

<whole molecular structure> ⇒ <one designated broken bond>

To achieve this transformation, INTSUM must hypothesize, for each fragment appearing in the mass spectrum, which bonds could have broken to produce that fragment. INTSUM employs a very simple theory of mass spectrometry (the so-called *half-order theory*) to propose these hypotheses. The result is one or more highly-specific cleavage rules for every fragment that appeared in any of the mass spectra in the original training instances.

These highly-specific cleavage rules are given to the second and third subprograms in the META-DENDRAL system: RULEGEN and RULEMOD. These two programs seek to find a small set of generalized cleavage rules that cover *most* of these highly-specific training rules. Notice that in this learning problem, no single generalized cleavage rule (or equivalently, no conjunctive generalization) can be expected to explain all of the training rules. In fact, since the INTSUM interpretation process can produce incorrect training instances, there is no reason to expect that even a set of cleavage rules will cover all of the training rules. Consequently, RULEGEN and RULEMOD do not search for MSC-generalizations. Instead, they develop a taxonomic description of the mass spectrometry data in the form of a *set* of cleavage rules that together cover the most important of the training rules.

The generalization process is done in two steps. First, RULEGEN conducts a model-driven generate-and-test search of the space of possible cleavage rules. This is a fairly coarse search from which redundant and approximate rules may result. The second phase of the search is conducted by the RULEMOD program, which cleans up the rules developed by RULEGEN to make them more precise and less redundant. We will concentrate on the description of the RULEGEN program,

since it employs a top-down, model-driven algorithm that can be compared, in part, to the other learning methods described in this chapter.

The RULEGEN algorithm chooses as its starting point the most general cleavage pattern (x ** y) with no properties specified for either atom. Since this pattern matches every bond in every molecule, it predicts that every bond will break. RULEGEN generates successively more refined rules by specializing this pattern. The algorithm performs a sort of breadth-first search. At each iteration (each level of the search tree) it specializes a parent cleavage pattern by making a change to all atoms at a specified distance (radius) from the ** bond—the bond designated to break. The change can involve either adding new neighbor atoms or specifying an atom feature. All possible specializations are made for which there are supporting training instances. The technique of modifying *all* atoms at a particular radius causes the RULEGEN search to be coarse.

After each cycle of specialization, the resulting bond patterns are tested against the training instances, and a heuristic measure of "improvement" is computed that indicates whether a newly specialized bond pattern is more plausible than its parent pattern. If a pattern is determined to be an improvement, it is retained, and the specialization process continues. If all specializations of a parent pattern are less plausible than their parent, the parent pattern is output as a new cleavage rule, and no more specializations of that pattern are considered.

The improvement criterion states that a child pattern graph is more plausible than its parent if:

- It predicts fewer fragmentations per training molecule (that is, it is more specific).
- It still predicts fragmentations for at least half of all of the training molecules (that is, it is sufficiently general).
- It predicts fragmentations for as many molecules as its parent—unless the parent graph was "too general" in the sense that the parent predicts more than 2 fragmentations in some single training molecule or on the average it predicts more than 1.5 fragmentations per molecule.

Thus, RULEGEN can be viewed as following paths of increasing specialization through the space of possible bond patterns until the improvement criterion achieves a local maximum. The result of this process is a set of such plausible bond patterns. RULEMOD improves this set by performing detailed hill-climbing searches in the region immediately around each generated bond pattern. For the detailed searches, negative training instances are employed as part of the plausibility criterion. Negative training instances are bond patterns for which the actual spectrum shows that the designated bond did *not* break. RULEMOD also compares the generated bond patterns with one another and removes bond patterns that are redundant. The result of the RULEMOD processing is a smaller set of more precise bond patterns. Each of these bond patterns is converted into a cleavage rule and printed out.

Evaluation:

It is somewhat difficult to compare META-DENDRAL to the other methods described in this chapter since it is such a complex system and since even the RULEGEN subprogram is not searching for MSG-generalizations. However, we have included META-DENDRAL because it is such an important and powerful learning system.

1. *Representational adequacy*. The bond-pattern representation was adequate for the task of developing cleavage rules. It was specifically designed for use in chemical domains and is not general. The descriptions can be viewed as conjunctions. Individual rules developed by the program can be considered to be linked by disjunction.

2. *Rules of generalization*. The dropping condition and turning constants to variables rules are used "in reverse" during the specialization process. META-DENDRAL also uses the generalization by internal disjunction rule. For example, it can learn that the number of non-hydrogen neighbors (*nbrs*) of an atom is "greater-than one." In related work on nuclear magnetic resonance (NMR), Schwenzer and Mitchell [1977] present an example in which the value of *nhs* is listed as "greater than or equal to one" (which indicates an internal disjunction).

3. *Computational efficiency*. The comparison of computational efficiency is not provided for META-DENDRAL because it is not possible to hand simulate its operation on the sample problem of Figure 3-2. First of all, it is impossible to represent the sample problem as a chemical graph because the problem uses *two* different connecting relationships (*ontop* and *inside*) whereas META-DENDRAL only allows one (chemical bonding). Secondly, as mentioned above, the algorithm seeks a taxonomic—not characteristic—description of the input examples. Thirdly, the termination criteria for the RULEGEN algorithm are stated in purely chemical terms that have no counterpart in the domain of geometric figures. The current program is considered to be relatively inefficient [Buchanan *et al.*, 1976].

4. *Flexibility and extensibility*. META-DENDRAL has been extended to handle NMR spectra [Schwenzer & Mitchell, 1977]. The program works well in an error-laden environment. It uses domain-specific knowledge extensively. However, there is no strict separation between a general-purpose induction component and a special-purpose knowledge component. It is not clear whether the methods developed for META-DENDRAL could be easily applied to any non-chemical domain.

5. META-DENDRAL has extensive constructive induction facilities. In particular, program INTSUM performs sophisticated transformations of the input spectrum in order to develop the bond-environment descriptions. Unfortunately, this part of the program is highly procedural. None of the rules of constructive induction have been made explicit nor is there a general facility for accepting additional rules of constructive induction from the user. The user can alter some of the parameters of the half-order theory, however.

3.2.3.2 Michalski and Dietterich: Program INDUCE 1.2

Michalski and his collaborators have worked on many aspects of inductive learning. Most relevant here are works by Larson and Michalski [Larson & Michalski, 1977; Larson, 1977; Michalski, 980a]. These articles describe a general method (and program) for determining disjunctive structural descriptions that can also be used (somewhat inefficiently) to discover MSC-generalizations. The method presented here is different from previous work and is specially designed for finding MSC-generalizations.

The language used to describe the input events is VL_{21} [Michalski, 980a], an extension to first-order predicate logic (FOPL) that was developed specifically for use in inductive inference.[4] Each event is represented as a conjunction of *selectors*. A *selector* is a relational statement that typically contains a function or predicate descriptor (with variables as arguments) and a list of values that the descriptor may assume. For example, the selector [size(v1) = small, medium] asserts that the size of v1 may take the values small or medium. Another form of selector is an n-ary predicate in brackets, which is interpreted in the same way as in FOPL. For example, the selector [ontop(v1,v2)] asserts that object v1 is ontop of object v2. A conjunction of selectors is denoted by their concatenation. The events in Figure 3-1 are represented as:

E1: ∃v1, v2 [size(v1) = small][size(v2) = small] &
[shape(v1) = circle][shape(v2) = square][ontop(v1,v2)]

E2: ∃v1, v2, v3 [size(v1) = small][size(v2) = large] &
[size(v3) = small][shape(v1) = circle] &
[shape(v2) = square][shape(v3) = circle] &
[ontop(v1,v2)][inside(v3,v2)]

In this method, we attempt to accelerate the search for plausible generalizations by using techniques similar to those of hierarchical planning [Sacerdoti, 1973] . First, we separate all descriptors into two classes, unary and non-unary. We call the unary descriptors *attribute descriptors* since they are typically used to represent attributes such as size or shape. Non-unary descriptors are called *structure-specifying descriptors* since they are typically used to specify structural information (for example, relationships ontop and inside).

The basic idea of the method is to first search the description space that is defined by the structure-specifying descriptors. Once plausible generalizations are found in this abstract *structure-only space*, attribute descriptor space is searched to fill out the detailed generalizations. There are several advantages to this two-phase approach as compared to a standard search of the entire description space:

The first is representational. As we have seen above, it is usually necessary to use a graph (or equivalent data structure) to represent an event in a structural learning problem. This is due to the fact that a graph is the most compact way to represent binary relationships among n objects when the number of such relationships is substantially less than the $n(n-1)$ possible relationships (that is, when the relationship matrix is sparse). Thus, in our method, the structure-only events are represented as graphs. But once we have located plausible points in this structure-only space, we can continue the search in attribute space. Attribute (or unary) descriptors can be represented as vectors that are substantially more compact and more efficiently manipulated than graphs.

[4] A somewhat modified and generalized form of VL_{21}, called the annotated predicated calculus, is described in Chapter 4 of this book.

The second advantage of this hierarchical approach is computational. The task of comparing two graph structures is NP-complete. Any decrease in the size of these graph structures leads to large decreases in the cost of a graph comparison. Furthermore, we can confine graph comparisons to the first phase of the algorithm.

A third advantage of this approach is that we can take "large steps" during the search for plausible descriptions by conducting much of the search in a sparse, abstract space. This is similar in spirit to the coarse search employed in RULEGEN.

There are also several disadvantages to this approach. Firstly, no speedup will be obtained unless the learning problem uses both unary and non-unary descriptors. There are some learning problems in which attributes play almost no role at all. In such cases, the structure-only search space is the same as the complete search space, so no computational savings will be obtained. There are also learning problems that require only unary descriptors (as in [Hunt *et al.*, 1966]). These are not structural learning problems, and the structure-only space is empty.

A second disadvantage of this approach involves the problem of defining "plausible" descriptions in structure-only space. One fact that can be used is the following: If g is a MSC-generalization in structure-only space, then there exists a full description G, such that g is the structure-only portion of G and G is a MSC-generalization in the complete space.

Thus, if we find all MSC-generalizations of the input events in structure-only space, then we can use these to find MSC-generalizations in the complete space. However, we will not necessarily find all possible MSC-generalizations in this fashion, since there may exist MSC-generalizations in the complete space whose structure-only component is *not* maximally specific in structure-only space. To avoid this problem, the algorithm may accept less than maximally-specific generalizations in the structure-only space (that is, more general descriptions) and terminate the search using some problem-oriented knowledge.

Another difficulty concerns how to conduct the attribute search once plausible structure-only descriptions have been located. Our approach is to use each structure-only description to define a new attribute-only space into which all of the input events are translated. Unfortunately, an input event can be mapped to more than one attribute-only description as shown below. This complicates the search.

The algorithm searches structure-only space using a "beam search"—a form of best-first search in which a *set* of best candidate descriptions is maintained during the search (see [Rubin & Reddy, 1977]). First, all unary descriptors are removed from the input events (thus abstracting them into structure-only space). Then a random sample of these events is taken to form set B_0, the initial set of generalizations (the initial beam set). In each step, B_i is first pruned to a fixed sized *beam width* by removing unpromising generalizations. (Promise is determined by the application of the heuristic evaluation functions described below). Then B_i is checked to see if any of its generalizations covers all of the input examples. If any do, they are removed from B_i and placed in the set C of candidate conjunctive generalizations. Lastly, B_i is generalized to form B_{i+1} by taking each element of B_i and generalizing it in all possible ways by dropping single selectors. When the set of candidates C reaches a prespecified size, the search halts. The set C contains conjunctive generalizations of the input data, some of which are maximally specific. The size limit on C determines how deeply the algorithm searches.

The program allows the user to employ simultaneously several criteria for evaluating the promise of intermediate generalizations. These criteria are combined to form a lexicographic evaluation functional with tolerances [Michalski, 1973]. Some of the criteria presently included in the program are:

- maximize the number of input events covered by a generalization.
- maximize the number of selectors in a generalization.
- minimize the total "cost" of the descriptors in a generalization. Different descriptors can be given costs according to their difficulty of measurement and other domain-dependent properties.

The user creates the evaluation functional by selecting criteria from a list of available criteria and ordering them in decreasing order of importance. Each criterion is accompanied by a tolerance that specifies the allowed departure of the associated criterion from the optimum value (see [Michalski, 1973]).

Once the structure-only candidate set C has been built, each candidate generalization in C must be filled out by finding values for its attribute descriptors. Each candidate generalization g in C is used to define an attribute-only space that is then searched using a beam search technique similar to that used to search the structure-only space. The attribute-only space is defined as follows. Let $\{v_1, v_2, ..., v_k\}$ be the existentially quantified variables used in the candidate structure-only generalization g. The attribute-only space generated by g is the space of all $m \times k$-tuples consisting of the values of the m attributes describing the k objects denoted by the quantified variables $\{v_1, v_2, ..., v_k\}$. In cases where some of the m attributes are not applicable to some of the objects, the attribute-only space will be correspondingly smaller.

In order to search this space, all of the input events must first be translated into this attribute-only space. This is accomplished by matching g against all input events and extracting the attributes of the variables in the input events that

match $v_1, v_2, ..., v_k$ in g. The values of these attributes form a single $m \times k$-tuple. For example, if g = [ontop(v1, v2)] and the variables v1 and v2 have two attributes, size and shape, then the attribute-only space generated by g is the space of all 4-tuples of the form:

< size(v1), size(v2), shape(v1), shape(v2) >

Let E_1 be the following input event:

E_1: ∃p1, p2, p3 [ontop(p1, p2)][ontop(p2, p3)] &
[size(p1) = 1][size(p2) = 3][size(p3) = 5] &
[color(p1) = red][color(p2) = green][color(p3) = blue]

Then we can translate E_1 into this attribute-only space in two different ways—since g matches E_1 in two distinct ways.

When g is matched to E_1 so that v1 is matched with p1 and v2 with p2, the resulting attribute-only 4-tuple is:

< 1, 3, red, green >

When v1 is matched to p2 and v2 to p3, then the resulting event is:

< 3, 5, green, blue >

During the search of this attribute-only space, the goal is to find an MSC-generalization that covers at least one of these two translated events (and thus covers E_1). Such an MSC-generalization is in the form of an $m \times k$-tuple as above, except that each position in the tuple may contain a *set* of values of the corresponding attribute. This set of values is expressed by an internal disjunction in the final corresponding formula.

The beam search of attribute-only space is similar to the search of structure-only space. A random sample of events is selected and generalized step-by-step by extending the internal disjunctions in the events. The generalization process is guided by a means-ends analysis to detect relevant differences between the current generalizations and events that have not yet been covered. Heuristic criteria are used to prune the beam set to a fixed beam width. Candidate generalizations that cover all of the input events (that is, at least one of the attribute-only events translated from each input event) are removed from the beam set and added to the candidate set C′. Each candidate in C′ provides possible settings of the attribute descriptors that, when combined with the structure-specifying descriptors in g, produces an output conjunctive generalization G.

Among all conjunctive generalizations produced by this algorithm, there may be some that are not maximally specific. This occurs when the search of structure-only space is permitted to produce candidate structure-only generalizations that are not maximally specific. In most observed cases such candidate generalizations become maximally specific when their attribute descriptors are filled in during the second phase of the algorithm.

Evaluation:

1. *Representational adequacy*. Using only selective rules of generalization, the algorithm discovers, among others, the following generalizations of the events in Figure 3-2:

- ∃v1, v2 [ontop(v1,v2)] [size(v1) = medium]
  [shape(v1) = polygon] [texture(v1) = blank]
  [size(v2) = medium∨large] [shape(v2) = rectangle∨circle]

  There exist two objects (in each event), such that one is a blank, medium-sized polygon on top of the other, a medium or large circle or rectangle.
- ∃v1, v2 [ontop(v1,v2)] [size(v1) = medium]
  [shape(v1) = circle∨square∨rectangle] [size(v2) = large]
  [shape(v2) = box∨rectangle∨ellipse] [texture(v2) = blank]

  There exist two objects such that one of them is a medium-sized circle, rectangle, or square on top of the other, a large, blank box, rectangle, or ellipse.
- ∃v1, v2 [ontop(v1,v2)] [size(v1) = medium] [shape(v1) = polygon]
  [size(v2) = medium∨large] [shape(v2) = rectangle∨ellipse∨circle]
  v

  There exist two objects such that one of them is a medium-sized polygon on top of the other, a large or medium rectangle, ellipse, or circle.
- ∃v1 [size(v1) = small∨medium]
  [shape(v1) = circle∨rectangle] [texture(v1) = shaded]

  There exists one object, a medium or small shaded circle or rectangle.

A few simple constructive induction rules have been incorporated into the current implementation. These include rules that count the number of objects possessing certain characteristics and rules that locate the top-most and bottom-most parts of an object (or more generally, extremal elements in a linearly-ordered set defined by any transitive relation, such as *On-top*). Other constructive induction rules can be specified by the user. Using the built-in constructive induction rules, the program produces the following conjunctive generalization of the input events in Figure 3-2:

- [# v's = 3,4] [# v's with texture blank = 2] &
  ∃v1, v2 [top-most(v1)] [ontop(v1,v2)]
  [size(v1) = medium] [shape(v1) = polygon]
  [texture(v1) = clear] [size(v2) = medium, large]
  [shape(v2) = circle, rectangle]

  There are either three or four objects in each event. Exactly two of these objects are blank. The top-most object is a medium-sized, clear polygon and it is on top of a large or medium-sized circle or rectangle.

This algorithm implements the conjunction, disjunction, and internal disjunction operators. The representation distinguishes among descriptors, variables, and values. Descriptors are further divided into structure-specifying descriptors and attribute descriptors. The current method discriminates among three types of descriptors:

- nominal—which have unordered value sets
- linear—which have linearly ordered value sets
- structured—which have tree-ordered value sets

This variety of possible representational forms is intended to provide a better "fit" between the description language and any specific problem.

2. *Rules of generalization*. The algorithm uses all rules of generalization mentioned in Section 3.1.5 and also a few constructive induction rules. It does not implement the introducing exception specialization rule. The effect of the turning constants to variables rule is achieved as a special case of the generalization by internal disjunction rule.

3. *Computational efficiency*. The algorithm requires 28 comparisons and builds 13 rules during the search to develop the descriptions listed above. Four rules are retained so this gives an efficiency ratio of 4/13 or 30%.

4. *Flexibility and extensibility*. The algorithm can be modified to discover disjunctions by altering the termination criteria for the search of structure-only space to accept structure conjuncts that do not necessarily cover all of the input events. The same general two-phase approach can also be applied to problems of determining discriminant descriptions. (See papers by Larson and Michalski [1977], Larson [1977], Michalski [1975, 1980a,b] and Chapter 4 of this book.)

The algorithm has good noise immunity. Noise events can be discovered because the algorithm tends to place them in separate terms of a disjunction.

Domain-specific knowledge can be incorporated into the program by defining the types and domains of descriptors, specifying the structures of these domains, specifying certain simple production rules (for domain constraints on legal combinations of variables), specifying the evaluation functional, and by providing constructive induction rules. These forms of knowledge representation are not always convenient, however. Further work should provide other facilities for knowledge representation.

As mentioned above, this method does perform a few general kinds of constructive induction. The method provides mechanisms for adding more rules of constructive induction.

The comparison of the above methods in terms of the criteria of Section 3.2.1 is summarized in Table 3-2.

3.3 CONCLUSION

This chapter has discussed various aspects of inductive learning of structural descriptions and has presented several criteria for evaluating learning methods. These criteria have been applied to the evaluation of five selected methods for learning structural descriptions. The main features revealed by this analysis are:

Table 3-2: Comparison of different learning methods

Method:						
Criterion:	**Winston**	**Hayes-Roth**	**Vere**	**Buchanan *et al.***	**Michalski**
Intended application	Learning toy block concepts	General	General	Discovering mass spectrometry rules	General
Language	Semantic net	Parameterized structural representation	Quantifier-free FOPL	Chemical model	Variable-valued Logic system VL_{21}
Syntactic concepts	Nodes and links of many types	Case frames parameter case labels	Literals constants	Molecule graph attributes constants in value sets	Selectors descriptors variables value sets
Operators	&, exception	&	&	&, V, internal V	&, V, internal V
Generalization and specialization rules:					
Dropping condition?	Yes	Yes	Yes	Yes	Yes
Constants to variables?	Yes	Yes	Yes	Yes	Yes
Generalizing by internal V?	No	No	No	No	Yes
Climbing tree?	Yes	No	No	No	Yes
Closing intervals?	No	No	No	No	Yes
Introducing exceptions?	Yes	No	No	No	No
Efficiency:					
Graph comparisons	2	22	Complete algorithm not known	Not applicable	28
Graph generations during search	11	20		Not applicable	13
Ratio of output to total	1/11 = 9%	6/20 = 30%		Not applicable	4/13 = 30%
Extensibility:					
Applications	None	Speech analysis	None	Mass spectrometry, NMR	Soybean disease diagnosis
	No	No	Yes	Yes	Yes
Disjunctive forms?	Very low	Low	Probably good	Excellent	Very good
Noise immunity?	Yes, built-in to program	No	Yes	Yes, built-in to program	Yes
Domain knowledge?	Limited facility	No	No	Extensive problem-specific facility	Few general rules
Constructive induction?					

tional efficiency on the one hand, and flexibility and extensibility on the other. Bottom-up methods tend to be faster, but have lower noise immunity and less flexibility. Top-down methods have good noise immunity and can be easily modified to discover disjunctive and other forms of generalization. They do tend to be computationally more expensive.

- *The description language employed by a learning method is critically important*. A learning method that uses a language with little structure (that is, that has few operators and few types of operands) tends to be relatively efficient and easy to implement but may not be able to learn descriptions that are most useful in real-world applications. On the other hand, a method that uses a language that is too rich will lead to enormous implementation problems that will be detrimental to successful research in machine learning.
- *A significant problem in current research on inductive learning is that each research group is using a different notation and terminology*. This not only makes the exchange of research results difficult, but it also makes it hard for new researchers to enter the field. This chapter has attempted to develop a set of concepts and criteria that abstract from these differences in notation and terminology.

The analysis raises some important problems to be addressed in future research:

- *Further work on representations*. Present learning programs are limited by the kinds of operators and variable types they allow, and also by the forms of descriptions they can produce. Methods for handling additional operators, variable types, and forms of descriptions need to be designed and implemented. Rules of generalization corresponding to these operators, types, and forms should also be developed. Among the forms that are particularly desirable are hierarchical and related forms in which a name of one description is used to build other, more complex descriptions. Some initial work in this area has been done by Winston [1970, 1975], Cohen and Sammut [1978], and in the area of grammatical inference in general (for example, Biermann [1972]).
- *The Principle of Comprehensibility*. In applications where people will need to use the generalizations produced by a learning program, it is important that the learning method produce generalizations that are easy to understand and close to corresponding natural language descriptions. This means that the descriptions developed by an inductive method must be structured to take into consideration human information processing *limitations*. As a rough guideline, conjunctions should involve no more than three or four conditions, full descriptions should involve only two or three disjunctive terms, and there should be no more than two quantifiers in the description. Descriptions should correspond to single "chunks" of information. Hierarchically-structured descriptions may provide a way to meet these guidelines. For more details, see Chapter 4 of this book.

mation. Hierarchically-structured descriptions may provide a way to meet these guidelines. For more details, see Chapter 4 of this book.

- *Constructive induction.* The constructive induction techniques developed to date are very limited. New rules of constructive induction need to be identified and implemented. An important problem is the development of efficient mechanisms for guiding the process of constructive induction through the potentially immense space of possible derived descriptors.
- *Integration of problem-specific knowledge.* Further work should be done on the problem of when and how to use problem-specific knowledge in a general induction method. The use of typed variables is a good example of a general way to incorporate problem-specific knowledge.
- *Extension to discriminant and taxonomic descriptions.* Much work has been done on characteristic generalization. Discriminant and taxonomic descriptions are very important, especially in noisy environments. More work on this subject is needed.
- *User interface.* As AI learning programs become more powerful, their functions will become more opaque. Learning programs should provide explanation facilities for justifying their generalizations.
- *Handling errors and missing data.* Very little attention has been paid to the problem of developing methods that work well in noisy environments. There is need for research on methods of learning from uncertain input information, from incomplete information, and from information containing errors.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the National Science Foundation under grants MCS-79-06614 and MCS-82-05166. This chapter is a descendant and an extension of a paper that first appeared in IJCAI-79. Since that time, reviewers from the AI Journal, where the initial version of the paper was subsequently published, and the editors of this volume have carefully read and criticized this work. The authors are grateful for these comments and criticisms, which significantly helped to improve the presentation and the readability of this chapter.

REFERENCES

Biermann, A. and Feldman, J., *A survey of results in grammatical inference*, Academic Press, New York, 1972.

Buchanan, B. G. and Feigenbaum, E. A., "DENDRAL and Meta-DENDRAL: their applications dimension," *Artificial Intelligence*, Vol. 11, pp. 5-24, 1978.

Buchanan, B. G., Feigenbaum, E. A. and Lederberg, J., "A heuristic programming study of theory formation in sciences," *Proceedings of the Second International Joint Conference on Artificial Intelligence*, International Joint Conferences on Artificial Intelligence, London, pp. 40-48, 1971.

Buchanan, B. G., Smith, D. H., White, W. C., Gritter, R. J., Feigenbaum, E. A., Lederberg, J., and Djerassi, C., "Applications of artificial intelligence for chemical inference XXII. Automatic rule formation in mass spectrometry by means of the Meta-DENDRAL program," *Journal of the American Chemical Society*, Vol. 98, pp. 6168, 1976.

Cohen, B. L. and Sammut, C. A., "Pattern recognition and learning with a structural description language," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, IJCPR, Kyoto, Japan, pp. 394, 1978.

Dietterich, T. and Michalski, R., "Inductive Learning of Structural Descriptions," *Artificial* Intelligence, Vol. 16, 1981.

Hayes-Roth, F., "Collected papers on the learning and recognition of structured patterns", Technical Report technical report, Carnegie-Mellon Department of Computer Science, Pittsburgh, PA., May 1976.

Hayes-Roth, F., "Patterns of induction and associated knowledge acquisition algorithms", Technical Report, Carnegie-Mellon University, May 1976.

Hayes-Roth, F. and McDermott, J., "Knowledge acquisition from structural descriptions," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, IJCAI, Cambridge, Mass., pp. 356-362, August 1977.

Hayes-Roth, F. and McDermott, J., "An interference matching technique for inducing abstractions," *Communications of the ACM*, Vol. 21, No. 5, pp. 401-410, 1978.

Hunt, E. B., Marin, J. and Stone, P. T., *Experiments in Induction*, Academic Press, New York, 1966.

Iba, G. A., "Learning disjunctive concepts from examples," Master's thesis, M.I.T., Cambridge, Mass., 1979, (also AI memo 548).

Knapman, J., "A critical review of Winston's learning structural descriptions from examples," *AISB Quarterly*, Vol. 31, pp. 319-320, September 1978.

Larson, J., *Inductive inference in the variable-valued predicate logic system VL21: methodology and computer implementation*, Ph.D. dissertation, University of Illinois, Urbana, Illinois, May 1977.

Larson, J. and Michalski, R. S., "Inductive inference of VL decision rules," *Proceedings of the Workshop on Pattern Directed Inference Systems, SIGART Newsletter 63*, pp. 38-44, June 1977.

Lenat, D. B., *AM: an artificial intelligence approach to discovery in mathematics as heuristic search*, Ph.D. dissertation, Stanford University, Stanford, California, 1976.

Michalski, R. S., "Discovering classification rules using variable-valued logic system VL1," *Proceedings of the Third International Joint Conference on Artificial Intelligence*, IJCAI, pp. 162-172, 1973.

Michalski, R. S., "Discovering classification rules using variable-valued logic system VL1," *Proceedings of the Third International Joint Conference on Artificial Intelligence*, IJCAI, pp. 162-172, 1973a.

Michalski, R. S., "Variable-Valued Logic and its Applications to Pattern Recognition and Machine Learning," *Multiple-Valued Logic and Computer Science*, Rine, D. (Ed.), North-Holland, pp. 506-534, 1975a.

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, 1980a.

Michalski, R. S. and Chilausky, R. L., "Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis," *Policy Analysis and Information Systems*, Vol. 4, No. 2, pp. 125-160, June 1980, (Special issue on knowledge acquisition and induction).

Mitchell, T. M., "Version Spaces: A candidate elimination approach to rule learning," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, IJCAI, Cambridge, Mass., pp. 305-310, 1977.

Mitchell, T. M., *Version Spaces: An approach to concept learning*, Ph.D. dissertation, Stanford University, December 1978, (also Stanford CS report STAN-CS-78-711, HPP-79-2).

Plotkin, G. D., "A note on inductive generalization," *Machine Intelligence*, Meltzer, B. and Michie, D. (Eds.), Edinburgh University Press, Edinburgh, pp. 153-163, 1970.

Plotkin, G. D., "A further note on inductive generalization," *Machine Intelligence*, Meltzer, B. and Michie, D. (Eds.), Elsevier, Edinburgh, pp. 101-124, 1971.

Quinlan, J. R., "Discovering rules from large collections of examples: a case study," *Expert Systems in the Micro Electronic Age*, Michie, D. (Ed.), Edinburgh University Press, Edinburgh, 1979.

Quinlan, J. R., "Induction over large data bases", Technical Report Report HPP-79-14, Heuristic Programming Project, Stanford University, 1979.

Rubin, S. M., and Reddy, R., "The locus model of search and its use in image interpretation," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, IJCAI, pp. 590-595, 1977.

Sacerdoti, E., "Planning in a hierarchy of abstraction spaces," *Proceedings of the Third International Joint Conference on Artificial Intelligence*, IJCAI, pp. 412-422, 1973.

Schwenzer, G. M., and Mitchell, T. M., *Computer-assisted structure elucidation using automatically acquired carbon-13* NMR *rules*, American Chemical Society, 1977.

Soloway, E. M., *Learning interpretation + generalization: a case study in knowledge-directed learning*, Ph.D. dissertation, University of Massachusetts at Amherst, 1978, (Computer and Information Science Report COINS TR-78-13).

Stepp, R., "Learning without negative examples via variable-valued logic characterizations: the Uniclass inductive program AQ7UNI", Technical Report 982, Department of Computer Science, University of Illinois at Urbana-Champaign, July 1979.

Vere, S. A., "Induction of concepts in the predicate calculus," *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, IJCAI, Tbilisi, USSR, pp. 281-287, 1975.

Vere, S. A., "Induction of relational productions in the presence of background information," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, IJCAI, Cambridge, Mass., pp. 349-355, 1977.

Vere, S. A., "Inductive learning of relational productions," *Pattern-Directed Inference Systems*, Waterman, D. A. and Hayes-Roth, F. (Eds.), Academic Press, New York, 1978.

Vere, S. A., "Multilevel counterfactuals for generalizations of relational concepts and productions," *Artificial Intelligence*, Vol. 14, No. 2, pp. 138-164, September 1980.

Winston, P. H., "Learning structural descriptions from examples", Technical Report AI-TR-231, MIT, Cambridge, Mass., September 1970.

Winston, P. H., "Learning structural descriptions from examples," *The Psychology of Computer Vision*, Winston, P. H. (Ed.), McGraw Hill, New York, ch. 5, 1975, (Original version published as a Ph.D. dissertaition, at MIT AI Lab, September, 1970).