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.

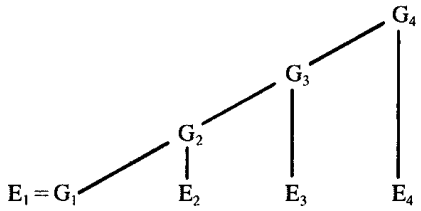
Larson's INDUCE-1 system [Larson, 1977; Larson & Michalski, 1977], on the other hand, performs constructive and selective induction simultaneously. INDUCE-1 implements the "find extrema of partial orders" rule of generalization described above, along with a few other constructive induction rules. New

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_I is the set of conjunctive generalizations obtained by taking each element of G_{I-1} and generalizing it with E_I.

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)

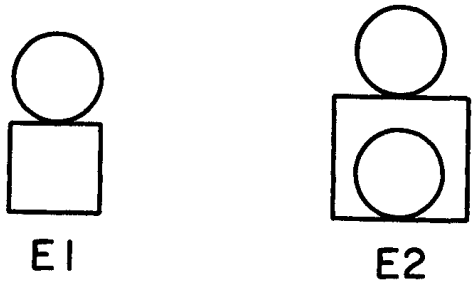


Figure 3-1: Sample problem for illustrating representation languages.

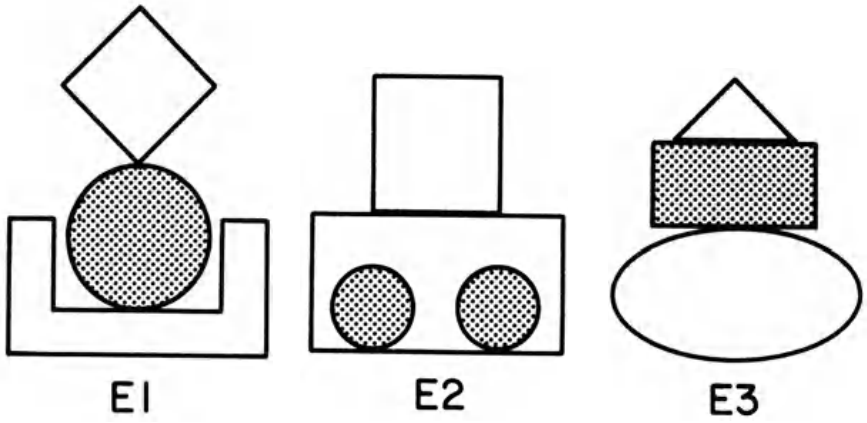


Figure 3-2: Sample problem for comparing the performance of the methods.

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.

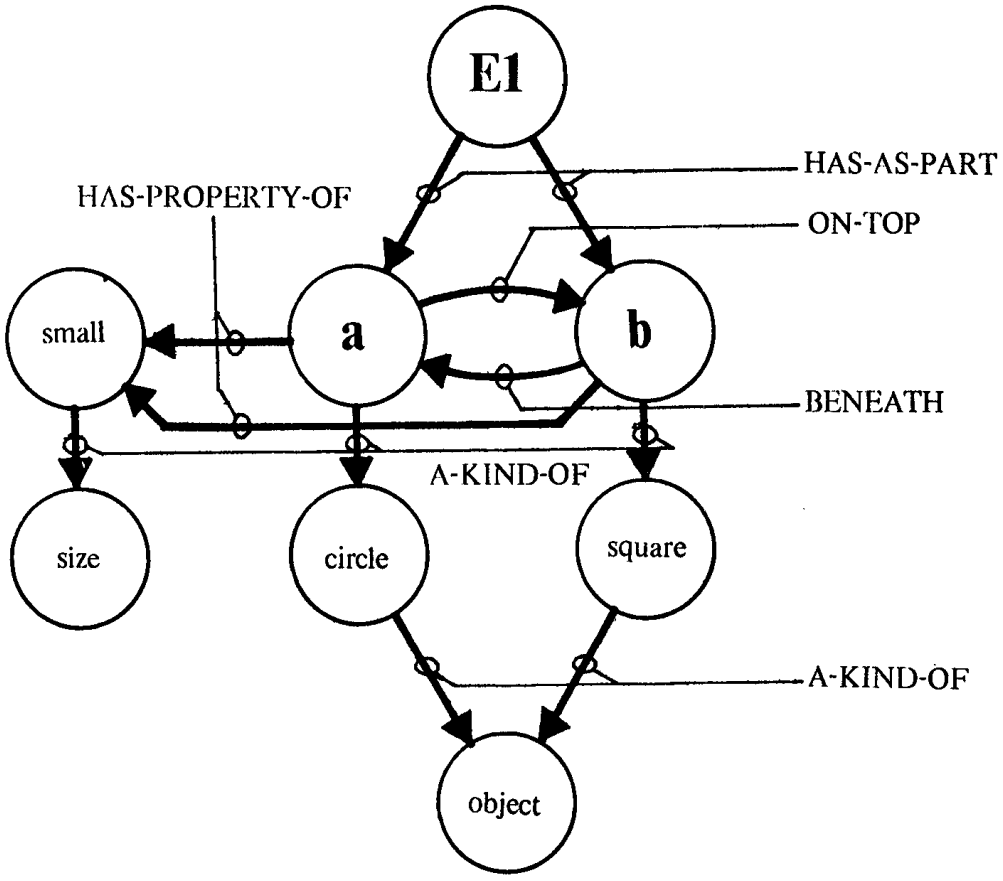


Figure 3-3: Network representing example E1 in Figure 3-1.

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

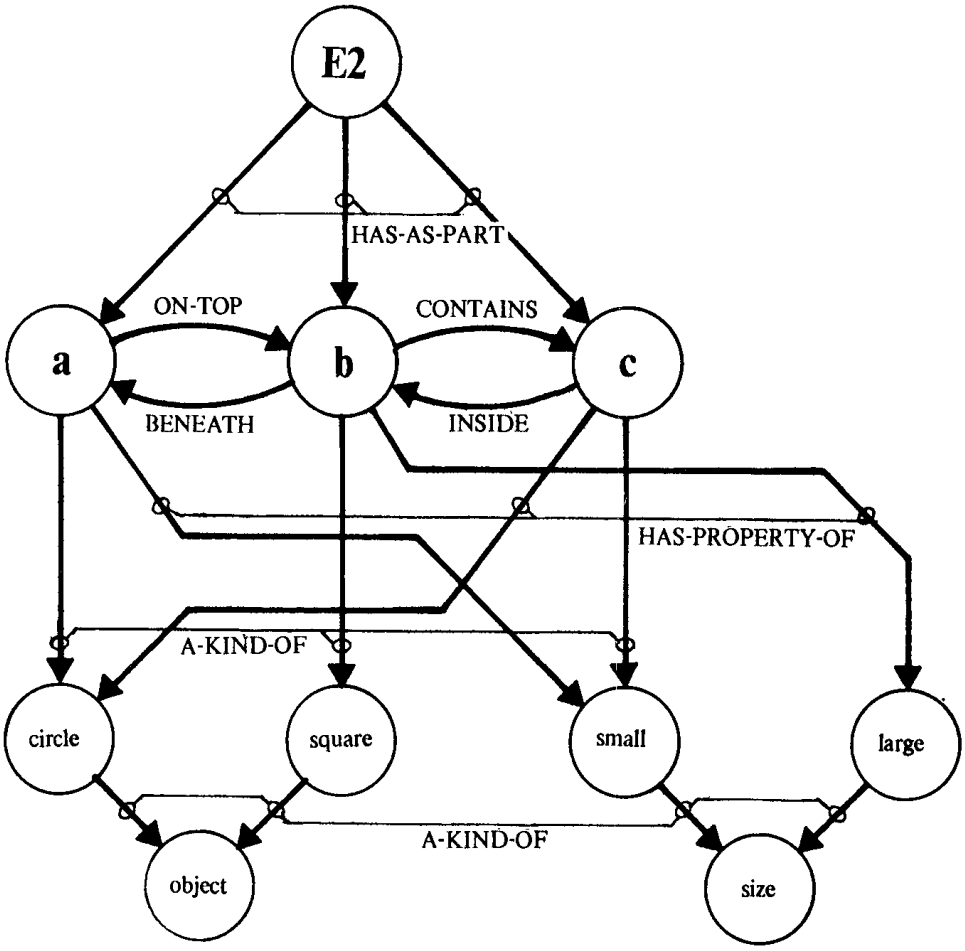


Figure 3-4: Network representing example E2 in Figure 3-1.

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].

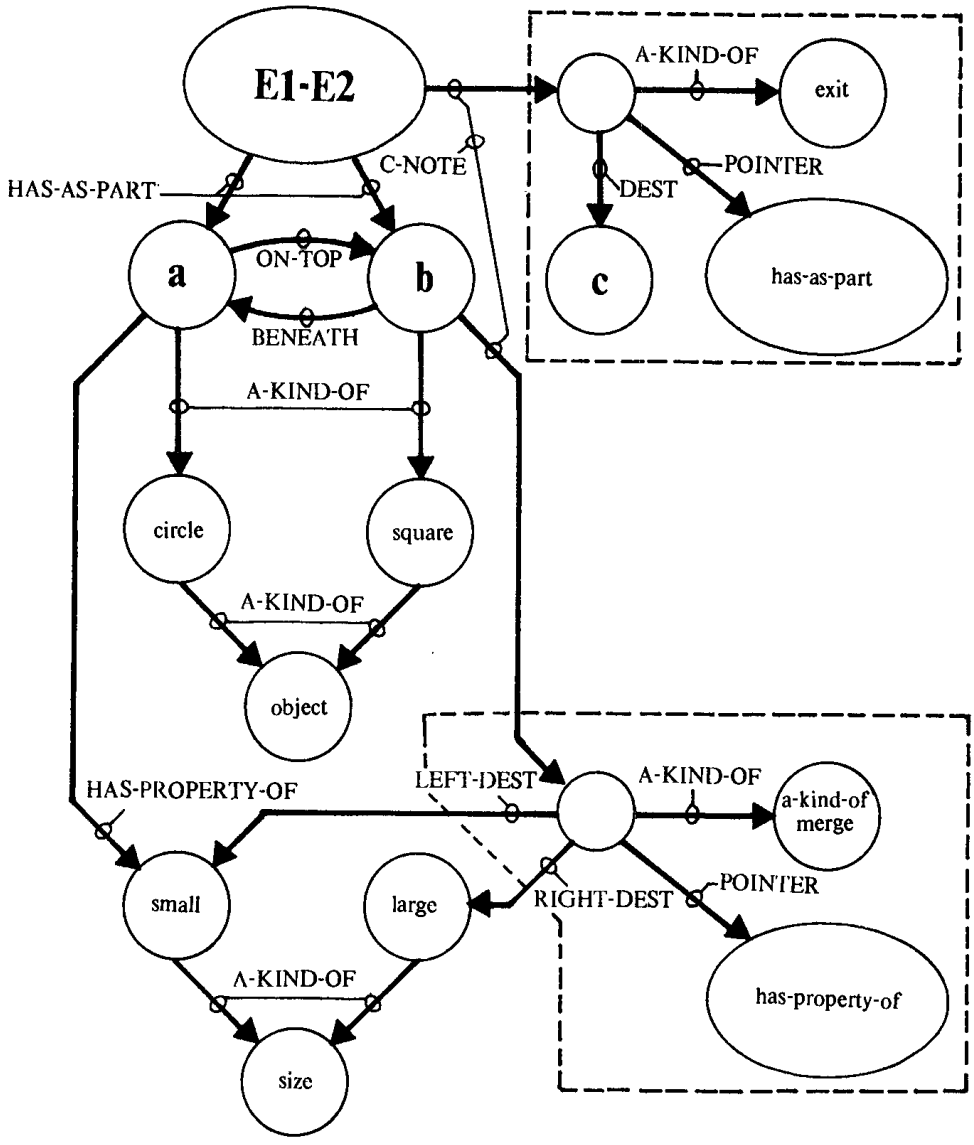


Figure 3-5: Difference description obtained by comparing E1 and E2 from Figure 3-1 and annotating the comparison with two C-NOTES.

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*.

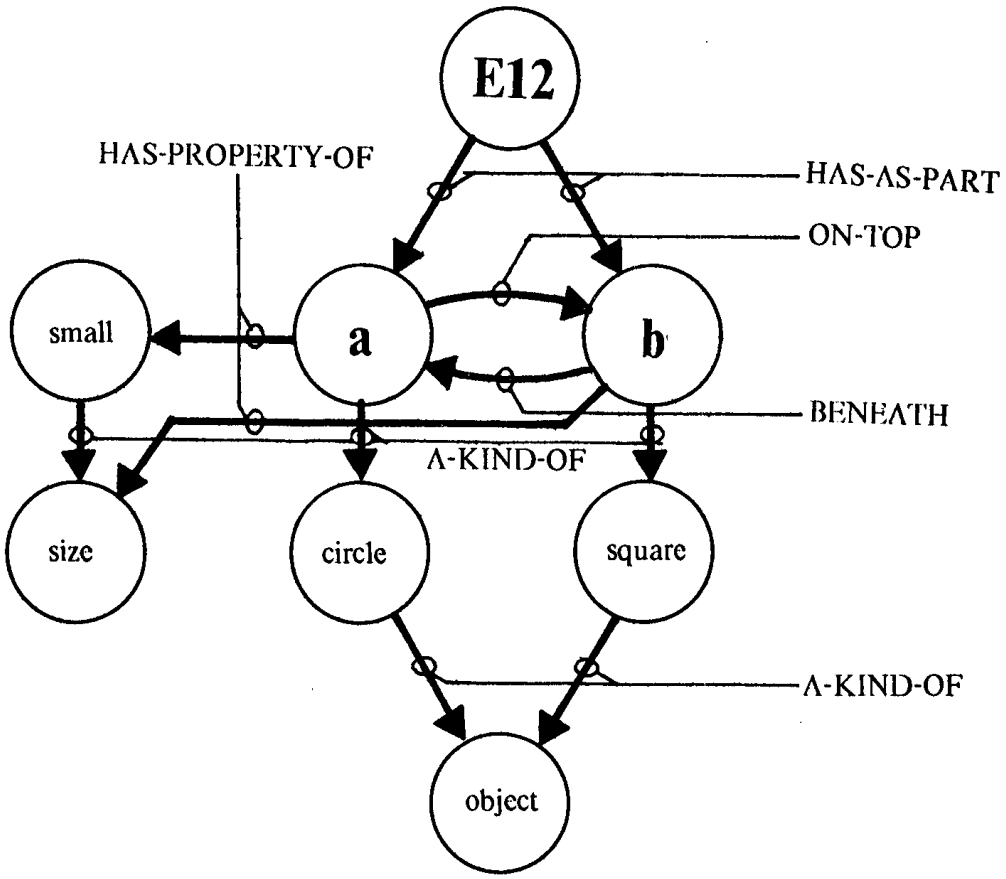


Figure 3-6: Network representing the generalized concept resulting from generalizing the difference description of Figure 3-5.

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.

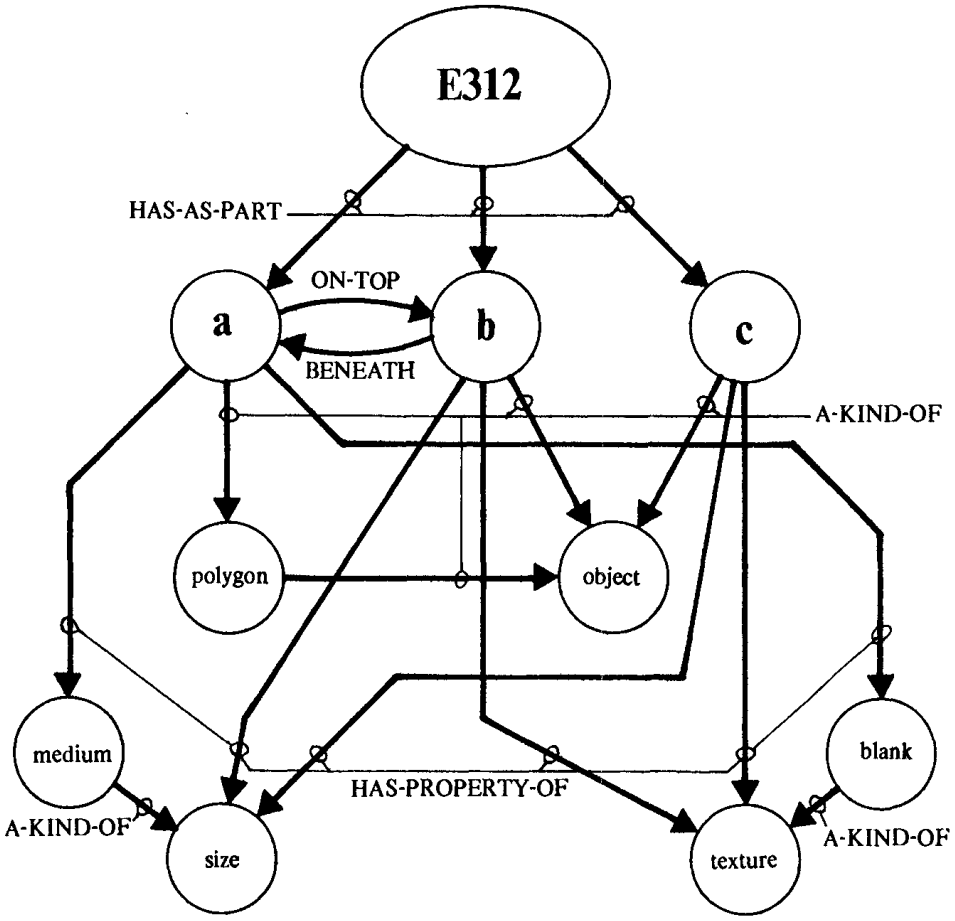


Figure 3-7: The first generalization obtained by simulating Winston's learning algorithm on the examples of Figure 3-2 (in the order E3, E1, E2). An English paraphrase is: "There is a medium, blank polygon on top of another object that has a size and texture. There is also another object with size and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

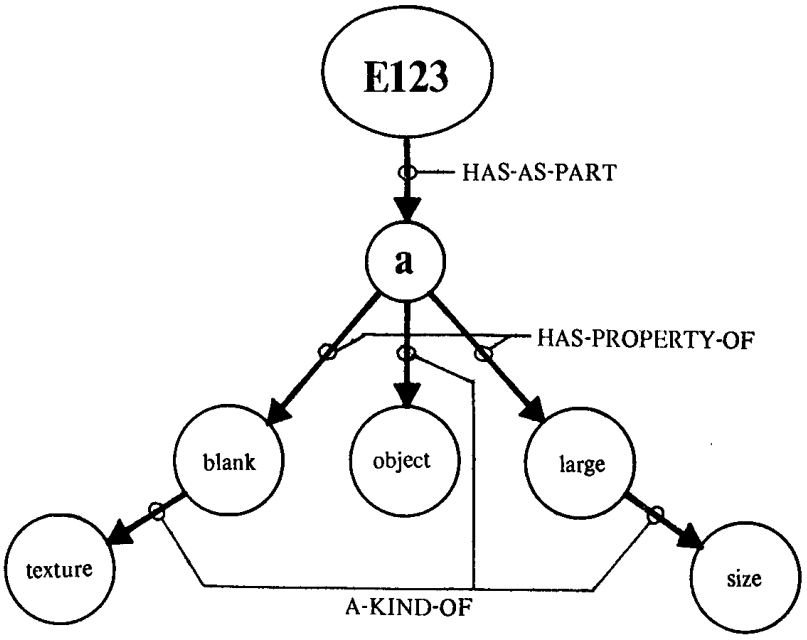


Figure 3-8: The second generalization obtained by simulating Winston's learning algorithm on the examples of Figure 3-2 (in the order E1, E2, E3). An English paraphrase is: "There is a large, blank object."

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}}