A THEORY AND METHODOLOGY OF INDUCTIVE LEARNING

by

Ryszard S. Michalski

Chapter in the Book, "Machine Learning: An Artificial Intelligence Approach," R. S. Michalski, J. Carbonell and T. Mitchell, (Editors), TIOGA Publishing Co., Palo Alto, California, 1983

A Theory and Methodology of Inductive Learning

Ryszard S. Michalski
Department of Computer Science, University of Illinois, Urbana, IL 61801, U.S.A.

Recommended Bruce Buchanan

ABSTRACT

A theory of inductive learning is presented that characterizes it as a heuristic search through a space of symbolic descriptions, generated by an application of certain inference rules to the initial observational statements (the teacher-provided examples of some concepts, or facts about a class of objects or a phenomenon). The inference rules include generalization rules, which perform generalizing transformations on descriptions, and conventional truth-preserving deductive rules (specialization and reformulation rules). The application of the inference rules to descriptions is constrained by problem background knowledge, and guided by criteria evaluating the 'quality' of generated inductive assertions.

Based on this theory, a general methodology for learning structural descriptions from examples, called STAR, *is described and illustrated by a problem from the area of conceptual data analysis.*

"... scientific knowledge through demonstration[1] is impossible unless a man knows the primary immediate premises ...," "... we must get to know the primary premises by induction; for the method by which even sense-perception implants the universal is inductive ..."
(circa 330 B.C.)
Aristotle, Posterior Analytics, Book II, Ch. 19.

1. Introduction

The ability of people to make accurate generalizations from few scattered facts or to discover patterns in seemingly chaotic collections of observations is a

[1] I.e., what we now call 'deduction'.

Artificial Intelligence **20** (1983) 111–161
0004-3702/83/0000–0000/$03.00

fascinating research topic of long-standing interest. The understanding of this ability is now also of growing practical importance, as it holds the key to an improvement of methods by which computers can acquire knowledge. A need for such an improvement is evidenced by the fact that knowledge acquisition is presently the most limiting 'bottleneck' in the development of modern knowledge-intensive artificial intelligence systems.

The above ability is achieved by a process called *inductive learning* i.e., inductive inference from facts provided by a teacher or the environment. The study and modeling of this form of learning is one of the central topics of machine learning. This paper outlines a theory of inductive learning and then presents a methodology for acquiring general concepts from examples.

Before going further into this topic, let us first discuss the potential for applications of inductive learning systems. One such application is an automated construction of knowledge bases for expert systems. The present approach to constructing knowledge bases involves a tedious process of formalizing experts' knowledge and encoding it in some knowledge representation system, such as production rules [75, 17] or a semantic network [7, 24]. Inductive learning programs could provide both an improvement of the current techniques and a basis for developing alternative knowledge acquisition methods.

In appropriately selected small domains, inductive programs are already able to determine decision rules by induction from examples of expert decisions. This process greatly simplifies the transfer of knowledge from an expert into a machine. The feasibility of such inductive knowledge acquisition has been demonstrated in the expert system PLANT/ds, for the diagnosis of soybean diseases. In this system, the diagnostic rules were developed in two ways: by formalizing experts' diagnostic processes and by induction from examples. In an experiment where both types of diagnostic rules were tested on a few hundred disease cases, the inductively derived rules outperformed the expert-derived ones [51]. Another example is an inductive acquisition of decision rules for a chess end-game [53, 61, 63].

A less direct, but potentially promising use of inductive learning is for the refinement of knowledge bases initially developed by human experts. Here, inductive learning programs could be used to detect and rectify inconsistencies, to remove redundancies, to cover gaps, and to simplify expert-derived decision rules. By applying an inductive inference program to the data consisting of original rules and examples of correct and incorrect results of these rules' application to new situations, the rules could be incrementally improved with little or no human assistance.

Another important application of inductive programs is in various experimental sciences, such as biology, chemistry, psychology, medicine, and

genetics. Here they could assist a user in detecting interesting conceptual patterns or in revealing structure in collections of observations. The widely used traditional mathematical and statistical data analysis techniques, such as regression analysis, numerical taxonomy, or factor analysis, are not sufficiently powerful for this task. Methods for *conceptual* data analysis are needed, whose results are not merely mathematical formulas but logic-style descriptions, characterizing data in terms of high-level, human-oriented concepts and relationships. An early example of such an application is the Meta-Dendral program [9] which infers cleavage rules for mass-spectrometer simulation (see its analysis in [20]).

There are two basic modes in which inductive programs can be utilized: as interactive tools for acquisition of knowledge from specific facts or examples, or as parts of machine-learning systems. In the first mode, a user supplies learning examples and exercises strong control over the way the program is used (e.g., [51, 70]).

In the second mode, an inductive program is a component of an integrated learning system whose other components generate the needed learning examples [10]. Such examples—positive and negative—constitute the feedback from the system's attempts to perform a desired task. An example of the second mode is the learning system LEX for symbolic integration [57], where a 'generalizer' module performs inductive inference on instances provided by a 'critic' module.

From the viewpoint of applications, such as aiding the construction of expert systems or conceptual analysis of experimental data, the most relevant is *conceptual inductive learning*. We use this term to designate a type of inductive learning whose final products are symbolic descriptions expressed in high-level, human-oriented terms and forms (more details are given in Section 3.1). The descriptions typically apply to real world objects or phenomena, rather than abstract mathematical concepts or computations. This paper is concerned specifically with conceptual inductive learning.

The most frequently studied type of such learning is *concept learning from examples* (called also *concept acquisition*), whose task is to induce general descriptions of concepts from specific instances of these concepts. The early studies of this subject go back to the fifties, e.g., those by Hovland [33], Bruner, Goodnow and Austin [8], Newell, Shaw and Simon [60], Amarel [1], Feigenbaum [21], Kochen [38], Banerji [2], Hunt [34], Simon and Kotovsky [76], Hunt, Marin and Stone [35], Hájek, Havel and Chytil [26] and Bongard [6]. Among more recent contributions there are those, for instance, by Winston [87], Waterman [86], Michalski [45], Hayes-Roth [28], Simon and Lea [77], Stoffel [83], Vere [85], Larson [40], Larson and Michalski [41], Mitchell [56], Quinlan [70] and Moraga [58]. An important variant of concept learning from

examples is the *incremental concept refinement*, where the input information includes, in addition to the training examples, previously learned hypotheses, or human-provided initial hypotheses that may be partially incorrect or incomplete (e.g., [52]). The paper by Dietterich and Michalski [20] discusses various evaluation criteria and several methods for concept learning from examples.

Another type of conceptual inductive learning is *concept learning from observation* (or *descriptive generalization*), concerned with establishing new concepts or theories characterizing given facts. This area includes such topics as automated theory formation (e.g., [9, 42, 43]), discovery of relationships in data (e.g., [27, 66, 39]), or an automatic construction of taxonomies (e.g., [50, 54]). Differences between concept learning from examples and concept learning from observation are discussed in more detail in the next section.

Conceptual inductive learning has a strong cognitive science flavor. Its emphasis on inducing human-oriented, rather than machine-oriented descriptions, and its primary interest in nonmathematical domains distinguishes it from other types of inductive learning, such as grammatical inference and program synthesis. In grammatical inference, the task is to determine a formal grammar that can generate a given set of symbol strings (e.g., [80, 4, 89, 23]). In program synthesis the objective is to construct a computer program from I/O pairs or computational traces, or to transform a program from one form to another by applying correctness-preserving transformation rules (e.g., [74, 11, 5, 13, 36, 79, 64]). The final result of such learning is a computer program, in an assumed programming language, destined for machine rather than human 'consumption'. For example, the method of 'model inference' by Shapiro [73] constructs a PROLOG program characterizing a given set of mathematical facts.

Recent years have witnessed the development of a number of task-oriented inductive learning systems that have demonstrated an impressive performance in their specific domain of application. Major weaknesses, however, persist in much of the research in this area. Most systems lack generality and extensibility. The theoretical principles upon which they are built are rarely well explained. Lack of common terminology and an adequate formal theory makes it difficult to compare different learning methods.

In the following sections we formulate logical foundations of inductive learning, define various types of such learning, present inference rules for generalizing concept descriptions, and finally describe a general methodology, called STAR, for learning structural descriptions from examples. To improve the readability of this chapter, below is presented a table of used symbols. Appendix A gives the details of the description language used (the annotated predicate calculus).

1.1. Symbols and notation

$\sim$ negation,
& conjunction (logical product),

$\vee$ disjunction (logical sum),
$\Rightarrow$ implication,
$\Leftrightarrow$ logical equivalence,
$\leftrightarrow$ term rewriting,
$\veebar$ exception (symmetric difference),
F a set of facts (formally, a predicate that is true for all the facts),
H a hypothesis (an inductive assertion),
$\rhd$ specialization,
$\lhd$ generalization,
$\models$ reformulation,
$\exists v_i$ existential quantifier over v_i,
$\exists(I)v_i$ numerical quantifier over v_i (I is a set of integers),
$\forall v_i$ universal quantifier over v_i,
D_i a concept description,
K_i a predicate asserting a concept name (a class) of objects,
$::>$ the implication linking a concept description with a concept name,
e_i an event (a description of an object),
E_i a predicate that is true only for the training events of the concept,
x_i an attribute (zero or one argument descriptor),
LEF a lexicographic evaluation functional,
DOM(p) the domain of descriptor p.

2. Types of Inductive Learning

2.1. Inductive paradigm

As mentioned before, inductive learning is a process of acquiring knowledge by drawing inductive inferences from teacher- or environment-provided facts. Such a process involves operations of generalizing, transforming, correcting and refining knowledge representations. Although it is one of the most common forms of learning, it has one fundamental weakness: except for special cases, the acquired knowledge cannot, in principle, be completely validated. This predicament, observed early on by Scottish philosopher David Hume (18th century), is due to the fact that inductively acquired assertions are hypotheses with a potentially infinite number of consequences, while only a finite number of confirming tests can be performed.

Traditional inquiries into inductive inference have therefore dealt with questions of what are the best criteria for guiding the selection of inductive assertions, and how can these assertions be confirmed. These are difficult problems, permeating all scientific activities. The search for answers has turned inductive inference into a battlefield of philosophers and logicians. There was even doubt whether it would ever be possible to formalize inductive inference and perform it on a machine. For example, philosopher Karl Popper [68] believed that inductive inference requires an irrational element. Bertrand

Russell in "History of Western Philosphy" [71] stated: "... so far no method has been found which would make it possible to invent hypotheses by rule". George Polya [67] in his pioneering and now classic treatise on plausible inference (of which inductive inference is a special case) observed: "A person has a background, a machine has not; indeed, you can build a machine to draw demonstrative conclusions for you, but I think you can never build a machine that will draw plausible inferences".

The above pessimistic prospects are now being revised. With the development of modern computers and subsequent advances in artificial intelligence research, it is now possible to provide a machine with a significant amount of background information. Also, the problem of automating inductive inference can be simplified by concentrating on the subject of hypothesis generation, while ascribing to humans the question of how to adequately validate them. Some successful inductive inference systems have already been built and a body of knowledge is emerging about the nature of this inference. The rest of this section will analyze the logical basis for inductive inference, and then Section 5 will present various generalization rules, which can be viewed as inductive inference rules.

In contrast to deduction, the starting premises of induction are specific facts rather than general axioms. The goal of inference is to formulate plausible general assertions that explain the given facts and are able to predict new facts. In other words, inductive inference attempts to derive a complete and correct description of a given phenomenon from specific observations of that phenomenon or of parts of it. As we mentioned earlier, of the two aspects of inductive inference—the generation of plausible hypotheses and their validation (the establishment of their truth status)—only the first is of primary interest to inductive learning research. The problem of hypothesis validation, a subject of various philosophical inquiries (e.g., [12]) is considered to be of lesser importance, because it is assumed that the generated hypotheses are judged by human experts, and tested by known methods of deductive inference and statistics.

To understand the role of inductive inference in learning, let us consider several different ways in which a system can acquire a description of a class of objects (situations, decisions, etc.).

(1) By receiving the description from a teacher and incorporating it within the system's existing knowledge structures (e.g., [25]). This way is called 'learning by being told'.

(2) By inferring the description from characteristics of a superset of the object class. This way is called 'learning by deductive inference'.

(3) By modifying the description already possessed about a similar class of objects (e.g., [88]). This way is called 'learning by analogy'.

(4) By generalizing teacher-provided examples and counter-examples of objects from this class. This way is called 'learning from examples'.

(5) By experimenting, discovering regularities, formulating useful concepts and structuring observations about the objects. This way is called 'learning from observation' (or 'learning by discovery').

Although all of these ways, except for the first one, involve some amount of inductive inference, in the last two, i.e., in learning from examples and in learning from observation, this inference is the central operation. These two forms are therefore considered to be the major forms of inductive learning. In order to explain them, let us formulate a general paradigm for inductive inference.

Given:

(a) *observational statements* (*facts*), F, that represent specific knowledge about some objects, situations, processes, etc.,

(b) a *tentative inductive assertion* (which may be null),

(c) *background knowledge* that defines the assumptions and constraints imposed on the observational statements and generated candidate inductive assertions, and any relevant problem domain knowledge. The last includes the *preference criterion* characterizing the desirable properties of the sought inductive assertion.

Find:

An *inductive assertion* (*hypothesis*), H, that tautologically or weakly implies the observational statements, and satisfies the background knowledge.

A hypothesis H tautologically implies facts F if F is a logical consequence of H, i.e., if the expression $H \Rightarrow F$ is true under all interpretations ('$\Rightarrow$' denotes logical implication). This is expressed as follows.

$$H \vdash F \quad \text{(read: } H \textit{ specializes to } F\text{)} \tag{1}$$

or

$$F \dashv H \quad \text{(read: } F \textit{ generalizes to } H\text{)}. \tag{2}$$

Symbols $\vdash$ and $\dashv$ are called the *specialization* and *generalization* symbols, respectively. If $H \Rightarrow F$ is valid, and H is true, then by the law of detachment (modus ponens) F must be true. Deriving F from H (deductive inference), is, therefore, truth-preserving. In contrast, deriving H from F (inductive inference) is not truth-preserving, but falsity-preserving, i.e., if some facts falsify F then they also must falsify H. (More explanation on this topic is given in Section 5.)

The condition that H *weakly implies* F means that facts F are not certain but only plausible or partial consequences of H. By allowing weak implication, this paradigm includes methods for generating 'soft' hypotheses, which hold only probabilistically, and partial hypotheses, which account for some but not all of the facts (e.g., hypotheses representing 'dominant patterns' or characterizing inconsistent data). In the following we will limit our attention to hypotheses that tautologically imply facts.

For any given set of facts, a potentially infinite number of hypotheses can be generated that imply these facts. Background knowledge is therefore necessary to provide the constraints and a preference criterion for reducing the infinite choice to one hypothesis or a few most preferable ones.

A typical way of defining such a criterion is to specify the preferable properties of the hypothesis—for example, to require that the hypothesis is the shortest or the most economical description consistent with all the facts (as, e.g., in [46]). Such a 'biased choice' criterion is necessary when the description language is complete, i.e., able to express any possible hypothesis. An alternative is to use a 'biased language' criterion [57], restricting the description language in which hypotheses are expressed (i.e., using an incomplete description language). Although in many methods the background knowledge is not explicitly stated, the authors make implicit assumptions serving the same purpose. More details on the criteria for selecting hypotheses are given in Section 4.7.

2.2. Concept acquisition vs. descriptive generalization

As mentioned in the introduction, one can distinguish between two major types of inductive learning: *learning from examples* (*concept acquisition*) and *learning from observation* (*descriptive generalization*). In concept acquisition, the observational statements are characterizations of some objects (situations, processes, etc.) preclassified by a teacher into one or more classes (concepts). The induced hypothesis can be viewed as a concept recognition rule, such that if an object satisfies this rule, then it represents the given concept. For example, a recognition rule for the concept 'philosopher' might be: 'A person who pursues wisdom and gains the knowledge of underlying reality by intellectual means and moral self-discipline is a philosopher'.

In descriptive generalization the goal is to determine a general description (a law, a theory) characterizing a collection of observations. For example, observing that philosophers Aristotle, Plato and Socrates were Greek, but that Spencer was British, one might conclude: 'Most philosophers were Greek'.

Thus, in contrast to concept acquisition that produces descriptions for classifying objects into classes on the basis of the objects' properties, descriptive generalization produces descriptions specifying properties of objects belonging to a certain class. Here are some example problems belonging to the above two categories:

Concept acquisition

(a) Learning *characteristic descriptions* of a class of objects, which specify one or more common properties of all known objects in the class. A logical product of all such properties defines the class in the context of an unlimited number of other object classes (e.g., [6, 87, 83, 85, 15, 29, 56, 82, 49, 20]).

(b) Learning *discriminant descriptions* of a class of objects that singly distinguish the given class from a limited number of other classes (e.g., [35, 6, 46–49, 70]).

(c) Inferring *sequence extrapolation rules* (e.g., [76, 19]) are able to predict the next element (a symbol, a number, an object, etc.) in a given sequence.

Descriptive generalization

(a) Formulating a theory characterizing a collection of entities (e.g., chemical compounds, as in [9], or numbers, as in [42, 43]).

(b) Discovering patterns in observational data (e.g., [26, 81, 27, 39, 66, 90]).

(c) Determining a taxonomic description (a classification) of a collection of objects (e.g., [49, 54]).

This paper is concerned primarily with problems of concept acquisition. In this case, the set of observational statements F can be viewed as a collection of implications:

$$F\colon \{e_{ik} :: > K_i\}, \quad i \in I, \tag{3}$$

where e_{ik} (a *training event*) denotes a description of the kth example of *concept* (*class*) asserted by predicate K_i (for short, *class* K_i) and I is a set indexing classes K_i. It is assumed here that any given event represents only one concept. Symbol $::>$ is used here, and will be used henceforth, to denote the implication linking a *concept description* with a predicate asserting the *concept name* (in order to distinguish this implication from the implication between arbitrary descriptions). The inductive assertion, H, can be characterized as a set of concept recognition rules:

$$H\colon \{D_i :: > K_i\}, \quad i \in I, \tag{4}$$

where D_i is a concept description of class K_i, i.e., an expression of conditions such that when they are satisfied by an object, then this object is considered an instance of class K_i.

According to the definition of inductive assertion, we must have

$$H \models F. \tag{5}$$

By substituting (3) and (4) for F and H, respectively, in (5), and making appropriate transformations, one can derive the following conditions to be satisfied in order that (5) holds

$$\forall i \in I\ (E_i \Rightarrow D_i) \tag{6}$$

and

$$\forall i, j \in I\ (D_i \Rightarrow \sim E_j), \quad \text{if } j \neq i, \tag{7}$$

where E_i, $i \in I$, is a description satisfied by all training events of class K_i, and only by such events (the logical disjunction of training events).

Expression (6) is called the *completeness condition*, and (7) the *consistency*

condition. These two conditions are the requirements that must be satisfied for an inductive assertion to be acceptable as a concept recognition rule. The completeness condition states that every training event of some class must satisfy the description D_i of the same class (since the opposite does not have to hold, D_i is equivalent to or more general than E_i). The consistency condition states that if an event satisfies a description of some class, then it cannot be a member of a training set of any other class. In learning a concept from examples and counterexamples, the latter constitute the 'other' class.

The completeness and consistency conditions provide the logical foundation of algorithms for concept learning from examples. We will see in Section 4 that to derive D_i satisfying the completeness condition one can adopt some inference rules of formal logic.

2.3. Characteristic vs. discriminant descriptions

The completeness and consistency conditions allow us to clearly explain the distinction between the previously mentioned characteristic and discriminant descriptions. A characteristic description of a class of objects (also known as *conjunctive generalization*) is an expression that satisfies the completeness condition or is the logical product of such expressions. It is typically a conjunction of some simple properties common to all objects in the class. From the applications viewpoint, the most interesting are *maximal characteristic descriptions* (maximal conjunctive generalizations or MCG) that are the most specific (i.e., longest) logical products characterizing all objects in the given class, using terms of the given language. Such descriptions are intended to discriminate the given class from all other possible classes (for illustration see Section 7).

A discriminant description is an expression that satisfies the completeness and consistency condition, or is the logical disjunction of such expressions. It specifies a single way or various alternative ways to distinguish the given class from a fixed number of other classes. The most interesting are *minimal discriminant descriptions* that are the shortest (i.e., with the minimum number of descriptors) expressions distinguishing all objects in the given class from objects of the other classes. Such descriptions are intended to specify the minimum information sufficient to identify the given class among a fixed number of other classes (for illustration see Section 7).

2.4. Single- vs. multiple-concept learning

It is instructive to distinguish between learning a single concept, and learning a collection of concepts. In *single-concept learning*, one can distinguish two cases: (1) when observational statements are just examples of the concept to be learned (learning from 'positive' instances only), and (2) when they are examples and counter-examples of the concept (learning from 'positive' and 'negative' instances).

In the first case, because of the lack of counter-examples, the consistency condition (7) is not applicable, and there is no natural limit to which description D_i (here, $i = 1$) can be generalized. One way to impose such a limit is to specify restrictions on the form and properties of the sought description. For example, one may require that it be the longest (most specific) conjunctive statement satisfying the completness condition (e.g., [85, 29]). Another way is to require that the description not exceed a given degree of generality, measured, for example, by the ratio of the number of all distinct events which could potentially satisfy the description to the number of training instances [82].

In the second case, when the teacher also provides counter-examples of the given concept, the learning process is considerably simplified. These counter-examples can be viewed as representing a 'different class', and the consistency condition (7) provides an obvious limit on the extent to which the hypothesis can be generalized. The most useful counter-examples are the so-called 'near misses' that only slightly differ from positive examples [87, 88]. Such examples place stronger constraints on the generalization process than randomly generated examples.

In *multiple-concept learning* one can also distinguish two cases: (1) when descriptions D_i of different classes are required to be mutually disjoint, i.e., no event can satisfy more than one description, and (2) when they are overlapping. In an overlapping generalization an event may satisfy more than one description. In some situations this is desirable. For example, if a patient has two diseases, his symptoms should satisfy the descriptions of both diseases, and in this case the consistency condition is not applicable.

An overlapping generalization can be interpreted in such a way that it always indicates only one decision class. In this special case, the concept recognition rules, $D_i :: > K_i$, are applied in a linear order, and the first rule satisfied generates the decision. In this case, if a concept description D_i for class K_i contains a conjunctively linked condition A, and precedes the rule for class K_j that contains condition $\sim A$, then the condition $\sim A$ is superflous and can be removed. As a result, the linearly ordered recognition rules can be significantly simplified. For example, the set of linearly ordered rules

$$D_1 :: > K_1, \qquad D_2 :: > K_2, \qquad D_3 :: > K_3$$

is logically equivalent to the set of (unordered) rules

$$D_1 :: > K_1, \qquad \sim D_1 \& D_2 :: > K_2, \qquad \sim D_1 \& \sim D_2 \& D_3 :: > K_3.$$

There are also other ways for deriving a single decision from overlapping rules (e.g., [17]). The above forms of multiple-concept learning have been implemented in inductive programs AQVAL/1 [46] and AQ11 [52].

3. Description Language

3.1. Bias toward comprehensibility

In concept acquisition, the main interest is in derivation of symbolic descriptions that are human-oriented, i.e., that are easy to understand and easy to use for creating mental models of the information they convey. A tentative criterion for judging inductive assertions from such a viewpoint is provided by the following *comprehensibility postulate.*

> The results of computer induction should be symbolic descriptions of given entities, semantically and structurally similar to those a human expert might produce observing the same entities. Components of these descriptions should be comprehensible as single 'chunks' of information, directly interpretable in natural language, and should relate quantitative and qualitative concepts in an integrated fashion.

As a practical guide, one can assume that the components of descriptions (single sentences, rules, labels on nodes in a hierarchy, etc.) should be expressions that contain only a few (say, less than five) conditions in a conjunction, few single conditions in a disjunction, at most one level of bracketing, at most one implication, no more than two quantifiers, and no recursion (the exact numbers may be disputed,[2] but the principle is clear). Sentences are kept within such limits by substituting names for appropriate subcomponents. Any operators used in descriptions should have a simple intuitive interpretation. Conceptually related sentences are organized into a simple data structure, preferably a shallow hierarchy or a linear list, such as a frame [55, 42].

The rationale behind this postulate is to ensure that descriptions generated by inductive inference bear similarity to human knowledge representations [31], and therefore, are easy to comprehend. This requirement is very important for many applications. For example, in developing knowledge bases for expert systems, it is important that human experts can easily and reliably verify the inductive assertions and relate them to their own domain knowledge. Satisfying the comprehensibility postulate will also facilitate debugging or improving the inductive programs themselves. When the complexity of problems undertaken by computer induction becomes very great, the comprehensibility of the generated descriptions will likely be a crucial criterion. This research orientation fits well within the role of artificial intelligence envisaged by Michie [44] to study and develop methods for man–machine conceptual interface and knowledge refinement.

[2]The numbers mentioned seem to apply to the majority of human descriptive sentences.

3.2. Language of assertions

One of the difficulties with inductive inference is its open-endedness. This means that when one makes an inductive assertion about some aspect of reality there is no natural limit to the level of detail in which this reality may be described, or to the richness of forms in which this assertion can be expressed. Consequently, when conducting research in this area, it is necessary to circumscribe very carefully the goals and the problem to be solved. This includes defining the language and the scope of allowed forms in which assertions will be expressed, as well as the modes of inference which will be used. The description language should be chosen so that crucial features are easily representable, while peripheral or irrelevant information are ignored.

An instructive criterion for classifying inductive learning methods is therefore the type of language used to express inductive assertions. Many authors use a restricted form of predicate calculus or closely related notation (e.g., [65, 22, 59, 85, 3, 50, 90, 72]). Some other formalisms include decision trees [35, 70], production rules (e.g., [86, 30]), semantic nets (e.g., [25]), and frames [42]. In earlier work (e.g., [45–48]) this author used a multiple-valued logic propositional calculus with typed variables, called VL_1 (the variable-valued logic system one). Later on an extension of the predicate calculus, called VL_2, was developed that was especially oriented to facilitate inductive inference [49].

Here we will use a somewhat modified and extended version of the latter language, to be called the *annotated predicate calculus* (APC). The APC adds to predicate calculus additional forms and new concepts that increase its expressive power and facilitate inductive inference. The major differences between the annotated predicate calculus and the conventional predicate calculus can be summarized as follows.

(1) Each predicate, variable and function (referred to collectively as a *descriptor*) is assigned an *annotation* that contains relevant problem-oriented information. The annotation may contain the definition of the concept represented by a descriptor, a characterization of its relationship to other concepts, a specification of the set over which the concept represented by a descriptor, a characterization of its relationship to other concepts, a specification of the set over which the descriptor ranges (when it is a variable or a function) and a characterization of the structure of this set, etc. (see Section 4).

(2) In addition to predicates, the APC also includes *compound predicates.* Arguments of such predicates can be *compound terms,* composed of two or more ordinary terms.

(3) Predicates that express relations $=$, $\neq$, $\geqslant$, $>$, $\leqslant$ and $<$ between terms or between compound terms are expressed explicitly as *relational statements,* also called *selectors.*

(4) In addition to the universal and existential quantifiers, there is also a

numerical quantifier that expresses quantitative information about the objects satisfying an expression.

The concept of annotation is explained more fully in the next section. Other aspects of the language are described in Appendix A. (The reader interested in a thorough understanding of this work is encouraged to read Appendix A at this point.)

4. Problem Background Knowledge

4.1. Basic components

As mentioned earlier, given a set of observational statements, one may construct a potentially infinite number of inductive assertions that imply these statements. It is therefore necessary to use some additional information, *problem background knowledge*, to constrain the space of possible inductive assertions and locate the most desirable one(s). In this section we shall look at various components of the problem background knowledge employed in the inductive learning methodology called STAR, described in Section 6. These components include the following.

- Information about descriptors (i.e., predicates, variables, and functions) used in observational statements. This information is provided by an *annotation* assigned to each descriptor (Section 4.3).
- Assumptions about the form of observational and inductive assertions.
- A preference criterion that specifies the desirable properties of inductive assertions sought.
- A variety of inference rules, heuristics, and specialized procedures, general and problem-dependent, that allow a learning system to generate logical consequences of given assertions and new descriptors.

Before we examine these components in greater detail, let us first consider the problem of how the choice of descriptors in the observational statements affects the generated inductive assertions.

4.2. Relevance of the initial descriptors

A fundamental problem underlying any machine inductive learning task is that of what information is provided to the machine and what information the machine is expected to produce or learn. As specified in the inductive paradigm, the major component of the input to a learning system is a set of observational statements. The descriptors used in those statements are observable characteristics and available measurements of objects under consideration. These descriptors are selected as relevant to the learning task by a teacher specifying the problem.

Determining these descriptors is a major part of any inductive learning problem. If they capture the essential properties of the objects, the role of the learning process is simply to arrange these descriptors into an expression

constituting an appropriate inductive assertion. If the selected descriptors are completely irrelevant to the learning task (as the color, weight, or shape of men in chess is irrelevant to deciding the right move), no learning system will be able to construct a meaningful inductive assertion.

There is a range of intermediate possibilities between the above two extremes. Consequently, learning methods can be characterized on the basis of the degree to which the initial descriptors are relevant to the learning problem.

Three cases can be distinguished.

(1) *Complete relevance.* In this case all descriptors in the observational statements are assumed to be directly relevant to the learning task. The task of the learning system is to formulate an inductive assertion that is a mathematical or logical expression of some assumed general form that properly relates these descriptors (e.g., a regression polynomial).

(2) *Partial relevance.* Observational statements may contain a large number of irrelevant or redundant descriptors. Some of the descriptors, however, are relevant. The task of the learning system is to select the most relevant ones and construct from them an appropriate inductive assertion.

(3) *Indirect relevance.* Observational statements may contain no directly relevant descriptors. However, among the initial descriptors there are some that can be used to construct derived descriptors that are directly relevant. The task of the learning system is to construct those derived descriptors and formulate an appropriate inductive assertion. A simple form of this case occurs, e.g., when a relevant descriptor is the volume of an object, but the observational statements contain only the information about the object's dimensions (and various irrelevant facts).

The above three cases represent problem statements that put progressively less demand on the relevance of the initial descriptors (i.e., that require less work from the person defining the problem) and more demand on the learning system. Early work on adaptive control systems and concept formation represents case (1). More recent research has dealt with case (2), which is addressed in *selective inductive learning.* A method of such learning must possess efficient mechanisms for determining combinations of descriptors that are relevant and sufficient for the learning task. Formal logic provides such mechanisms, and therefore it has become the major underlying formalism for selective methods.

An example of a selective learning method is the one implemented in program AQ11 [52] that inductively determined soybean disease diagnostic rules for the system PLANT/ds, mentioned in the introduction. A different type of selective method was implemented in program ID3 [70] that determines a decision tree for classifying a large number of events. A comparison between these two programs is described by O'Rorke [63].

Case (3) represents the task of *constructive inductive learning.* Here, a method must be capable of formulating new descriptors (i.e., new concepts, new variables, etc.), of evaluating their relevance to the learning task, and of

using them to construct inductive assertions. There has been relatively little done in this area. The 'automated mathematician' program AM [42] can be classified as a domain-specific system of this category. Some constructive learning capabilities have been incorporated in system BACON that automatically formulates mathematical expressions encapsulating chemical and other laws [39]. The general-purpose INDUCE program for learning structural descriptions from examples has implemented several general purpose constructive generalization techniques [40, 49]. Section 5 and 6 give more details on this subject.

4.3. Annotation of descriptors

An *annotation* of a descriptor (i.e., of a predicate, a variable or a function) is a store of background information about this descriptor tailored to the learning problem under consideration. It may include:
- a specification of the *domain* and the *type* of the descriptor (see below),
- a specification of operators applicable to it,
- a specification of the constraints and the relationships between the descriptor and other descriptors,
- for numerical descriptors, the mean, the variance, or the complete probability distribution of values for the problem under consideration,
- a characterization of objects to which the descriptor is applicable (i.e., a characterization of its possible arguments),
- a specification of a descriptor class containing the given descriptor that is the parent node in a generalization hierarchy of descriptors (for example, for descriptors 'length', 'width' and 'height', the parent node would be the 'dimensions'),
- synonyms that can be used to denote the descriptor,
- a definition of a descriptor (when it is derived from some other descriptors),
- if a descriptor denotes a class of objects, typical examples of this class can be specified.

Let us consider some of the above components of the annotation in greater detail.

4.4. The domain and type of a descriptor

Given a specific problem, it is usually possible to specify the set of values each descriptor could potentially adopt in characterizing any object in the population under consideration. Such a set is called the *domain* (or the *value set*) of the descriptor. The domain is used to constrain the extent to which a descriptor can be generalized. For example, the information that the temperature of a living human being may very, say, only between 34°C and 44°C prevents the system from considering inductive assertions in which the descriptor 'body temperature' would assume values beyond these limits.

Other important information for conducting the generalization process is concerned with the structure of the domain, that is, with the relationship existing among the elements of the domain. For numerical descriptors, such relationships are specified by the measurement scale. Depending on the structure of the descriptor domain, we distinguish among three basic types of descriptors.

(1) *Nominal (categorical) descriptors.* The value set of such descriptors consists of independent symbols or names, i.e., no structure is assumed to relate the values in the domain. For example, 'blood-type(person)' and 'name(person)' are unary nominal descriptors. Predicates, i.e., descriptors with the value set {True, False}, and n-ary functions whose ranges are unordered sets, are also nominal descriptors. An example of a two-argument nominal descriptor is 'license-plate-number(car, owner)', which denotes a function assigning to a specific car of the given owner a license plate number.

(2) *Linear descriptors.* The value set of linear descriptors is a totally ordered set. For example, a person's military rank or the temperature, weight or number of items in a set is such a descriptor. Variables measured on ordinal, interval, ratio, and absolute scales are special cases of a linear descriptor. Functions that map a set into a totally ordered set are also linear descriptors, e.g., 'distance(P_1, P_2)'.

(3) *Structured descriptors.* The value set of such descriptors has a tree or oriented graph structure that reflects the generalization relation between the values, i.e., is a generalization hierarchy. A parent node in such a structure represents a more general concept than the concepts represented by its children nodes. For example, in the value set of descriptor 'place', 'U.S.A.' would be a parent node of the nodes 'Indiana', 'Illinois', 'Iowa', etc. The domain of structured descriptors is defined by a set of inference rules specified in the problem background knowledge (see, e.g., descriptor 'shape(B_i)', in Section 6).

Structured descriptors can be further subdivided into ordered and unordered structured descriptors. Sometimes descriptors themselves can also be organized into a generalization hierarchy. For example, descriptors' length, width, and depth belong to a class of 'dimensions'. Information about the type of a descriptor is useful, as it determines the operations applicable to a descriptor.

4.5. Constraints on the description space

For a given induction problem there may exist a variety of constraints on the space of the acceptable concept descriptions, due to the specific properties and relationships among descriptors. Here are a few examples of such relationships.

- *Interdependence among values.* In many practical problems some variables specify a state of an object, and some other variables characterize the state. Depending on the values of the state-specifying variables, the variables charac-

terizing a state may be needed or not. For example, if a descriptor 'state(plant's leaf)' takes on value 'diseased', then a descriptor 'leaf discoloration' will be used to characterize the change of the leaf's color. When the descriptor 'state(plant's leaf)' takes on value 'normal', then obviously the 'leaf discoloration' descriptor is irrelevant. Such information can be represented by an implication:

$$[\text{state(plant's leaf)} = \text{normal}] \Rightarrow [\text{discoloration(plant's leaf)} = \text{NA}],$$

where NA is a special value meaning 'not applicable'.

- *Properties of descriptors.* Descriptors that are relations between objects may have certain general properties—they can be reflexive, symmetric, transitive, etc. All such properties are defined as assertions in the annotated predicate calculus (see Appendix A). For example, the transitivity of relation 'above(P_1,P_2)' can be defined as

$$\forall P_1, P_2, P_3, \quad (\text{above}(P_1, P_2)) \;\&\; \text{above}(P_2, P_3) \Rightarrow \text{above}(P_1, P_3).$$

- *Interrelationships among descriptors.* In some problems there may exist relationships between descriptors that constrain their values. For example, the length of an object is assumed always to be greater than or equal to its width:

$$\forall P, \quad \text{length}(P) \geq \text{width}(P).$$

Also, descriptors may be related by known equations. For example, the area of a rectangle is the arithmetic product of its length and width:

$$\forall P, \quad ([\text{shape}(P) = \text{rectangle}] \Rightarrow [\text{area}(P) = \text{length}(P) \cdot \text{width}(P)]).$$

(The infix operator '·' is used to simplify notation of the term multiply(length(P), width(P)).)

4.6. The form of observational and inductive assertions

The basic form of assertions in the STAR methodology is a *c-expression*, defined as a conjunctive statement:

$$\langle\text{quantifier form}\rangle\langle\text{conjunction of relational statements}\rangle, \tag{8}$$

where ⟨quantifier form⟩ stands for zero or more quantifiers, and ⟨relational statements⟩ are predicates in a special form, as defined in Appendix A. The following is an example of a c-expression:

$$\begin{aligned}\exists .P_0, P_1, P_2, P_3([\text{contains}(P_0, P_1, P_2, P_3)]\,[\text{ontop}(P_1 \,\&\, P_2, P_3)] \\ [\text{length}(P_1) = 3\,..\,8]|[\text{weight}(P_1) \geq \text{weight}(P_2)] \\ [\text{color}(P_1) = \text{red} \vee \text{blue}]\,[\text{shape}(P_1 \,\&\, P_2 \,\&\, P_3) = \text{box}])\end{aligned}$$

that can be paraphrased in English as follows.

> An object P_0 contains parts P_1, P_2 and P_3 and only these parts. Parts P_1 & P_2 are on top of part P_3, length of P_1 is between 3 and 8, the color of P_1 is red or blue, the weight of P_1 is greater than that of P_2, and the shape of all three parts is box.

An important special case of a c-expression is an *a-expression* (an *atomic expression*), in which there is no 'internal disjunction' (see Appendix A).

Note that due to the use of internal disjunction a c-expression represents a more general concept than a universally quantified conjunction of predicates, used in typical production rules.

Progressively more complex forms of expressions are described below.

- A *case expression* is a logical product of implications:

$$[L = a_i] \Rightarrow \mathrm{Exp}_i, \quad i = 1, 2, \ldots$$

where a_i are single elements or disjoint subsets of elements from the domain of descriptor L, and Exp_i are c-expressions.

A case expression describes a class of objects by splitting it into separate cases, each represented by a different value of a certain descriptor.

- An *implicative expression* (i-expression)

$$C \,\&\, (C_1 \Rightarrow C_2), \tag{9}$$

where C, C_1 and C_2 are c-expressions.

This form of description is very useful when the occurrence of some properties (defined in C_2) depends on the occurrence of some other properties (defined in C_1). Typical production rules used in expert systems are a special case of (9), where C is omitted and no internal logical operators are used. When $(C_1 \Rightarrow C_2)$ is omitted, then the conditional expression becomes a c-expression.

- A *disjunctive expression* (d-expression) defined as a disjunction of implicative expressions.

- An *exception-based expression* (e-expression).

In some situations it is simpler to formulate a somewhat overgeneralized statement and indicate exceptions than to formulate a precise statement. The following form is used for such purposes:

$$D_1 \vee D_2$$

where D_1 and D_2 are d-expressions. This expression is equivalent to $(\sim D_2 \Rightarrow D_1)$ & $(D_2 \Rightarrow \sim D_1)$.

Observational assertions are formulated as a set of rules

$$\{\text{a-expression} : :> K_i\}. \tag{10}$$

Inductive assertions are expressed as a set of rules

$$\{\text{EXP} :: > \text{c-expression}\}, \tag{11}$$

where EXP is a c-expression or any of the more complex expressions described above. It is also assumed that the left side and the right side of (11) satisfy the principle of comprehensibility described in Section 2.

4.7. The preference criterion

In spite of the constraints imposed by the above components of the background knowledge, the number of inductive assertions consistent with observational statements may still be unlimited. The problem then arises of choosing the most desirable inductive assertion(s). In making such a choice one must take into consideration the aspects of the particular inductive learning problem, and therefore the definition of a 'preference criterion' for selecting a hypothesis is a part of the problem background knowledge. Typically, the inductive assertions are chosen on the basis of some simplicity criterion (e.g., [37, 69]).

In the context of scientific discovery, philosopher Karl Popper [68] has advocated constructing hypotheses that are both simple and easy to refute. By generating such hypotheses and conducting experiments aimed at refuting them, he argues, one has the best chance of ultimately formulating the true hypothesis. In order to use this criterion for automated inductive inference it is necessary to define it formally. This, however, is not easy because there does not seem to exist any universal measure of hypothesis simplicity and refutability.

Among more specific measures for evaluating the 'quality' of inductive assertions one may list:

- An overall simplicity for human comprehension, measured, for example, by the number of descriptors and number of operators used in an inductive assertion.
- The degree of 'fit' between the inductive and observational assertions (measured, for example, by the degree of generalization, defined as the amount of uncertainty that any given description satisfying the inductive assertion corresponds to some observational statement [49]).
- The cost of measuring values of descriptors used in the inductive assertion.
- The computational cost of evaluating the inductive assertion.
- The memory required for storing the inductive assertion.
- The amount of information needed for encoding the assertion using a priori defined operators [16].

The importance given to each such measure depends on the ultimate purpose of constructing the inductive assertions. For that reason, the STAR methodology allows a user to build a global preference criterion as a function of such measures, tailored to a specific inductive problem. Since some of the above

measures are computationally costly, simpler measures are used, called *elementary criteria.* Among such criteria are the number of c-expressions in the assertion, the total number of relational statements, the ratio of possible but unseen events implied by an assertion to the total number of training events (a simple measure of generalization [50]), and the total number of different descriptors. The global preference criterion is formulated by selecting from the above list those elementary criteria that are most relevant to the problem, and then arranging them into a *lexicographic evaluation functional* (LEF). A LEF is defined as a sequence of criterion-tolerance pairs

$$\text{LEF: } (c_1, \tau_1), (c_2, \tau_2) \ldots, \tag{12}$$

where c_i is an elementary criterion selected from the available 'menu', and τ_1 is a *tolerance threshold* for criterion c_i ($\tau_i \in [0 \ldots 100\%]$).

Given a set of inductive assertions, the LEF determines the most preferable one(s) in the following way.

In the first step, all assertions are evaluated from the viewpoint of criterion c_1, and those which score best, or within the range defined by the threshold τ_1 from the best, are retained. Next, the retained assertions are evaluated from the viewpoint of criterion c_2 and reduced similarly as above, using tolerance τ_2. This process continues until either the subset of retained assertions contains only one assertion (the 'best' one) or the sequence of criterion-tolerance pairs is exhausted. In the latter case, the retained set contains assertions that are equivalent from the viewpoint of the LEF.

An important and somewhat surprising property of such an approach is that by properly defining the preference criterion, the same learning system can generate either characteristic or discriminant descriptions of object classes (see Section 7).

5. Generalization Rules

5.1. Definitions and an overview

Constructing in inductive assertion from observational statements can be conceptually characterized as a heuristic state-space search [62], where

- *states* are symbolic descriptions; the initial state is the set of observational statements;
- *operators* are inference rules, specifically, generalization, specialization and reformulation rules, as defined below;
- the *goal* state is an inductive assertion that implies the observational statements, satisfies the problem background knowledge and maximizes the given preference criterion.

A *generalization rule* is a transformation of a description into a more general description, one that tautologically implies the initial description. A *specializa-*

tion rule makes an opposite transformation: given a description, it generates a logical consequence of it. A *reformulation rule* transforms a description into another, logically equivalent description. A reformulation rule can be viewed as a special case of a generalization and a specialization rule.

Specialization and reformulation rules are the conventional truth-preserving inference rules used in deductive logic. In contrast to them, the generalization rules are not truth-preserving but falsity preserving. This means that if an event falsifies some description, then it also falsifies a more general description. This is immediately seen by observing that $H \Rightarrow F$ is equivalent to $\sim F \Rightarrow \sim H$ (the law of contraposition). To illustrate this point, suppose that a statement 'some water birds in this lake are swans' has been generalized to 'all water birds in this lake are swans'. If there are no water birds in the lake that are swans, then this fact falsifies not only the first statement but also the second. Falsifying the second statement, however, does not imply the falsification of the first.

In concept acquisition, as explained in Section 2, transforming a rule $E :: > K$ into a more general rule $D :: > K$ means that description E must imply description D:

$$E \Rightarrow D \tag{13}$$

(recall expression (6)). Thus, to obtain a generalization rule for concept acquisition, one may use a tautological implication of formal logic. The premise and consequence of such an implication must, however, be interpretable as a description of a class of objects. For example, the known law of simplification

$$P \,\&\, Q \Rightarrow P \tag{14}$$

can be turned into a generalization rule:

$$P \,\&\, Q :: > K \vdash P :: > K. \tag{15}$$

If P stands for 'round objects', Q for 'brown objects' and K for 'balls', then rule (15) states that the expression 'round and brown objects are balls' can be generalized to 'round objects are balls'. Thus, in concept acquisition, the generalization operation has a simple set-theoretical interpretation: a description is more general if it is satisfied by a larger number of objects. (Such an interpretation does not apply, however, to descriptive generalization, as shown below.)

In order to obtain a rule for descriptive generalization, implication (14) is reversed, and P and Q are interpreted as properties of objects of some class K

$$P(K) \vdash P(K) \,\&\, Q(K). \tag{16}$$

If $P(K)$ stands for 'balls are round' and $Q(K)$ for 'balls are brown', then

according to rule (16), the statement 'balls are round and brown' is a generalization of the statement 'balls are round' (because from the former one can deduce the latter). We can see that the notion 'the number of objects satisfying a description' is not applicable here. Generalizing means here adding (hypothesizing) properties that are ascribed to a class of objects.

After this informal introduction we shall now present various types of generalization rules, concentrating primarily on the rules for concept acquisition. These rules will be expressed using the notation of the annotated predicate calculus (see Appendix A). The reverse of these rules are specialization rules or reformulation rules in special cases. With regard to other specialization and reformulation rules we shall refer the reader to a standard book on predicate calculus (e.g., [84]). Some reformulation rules of the annotated predicate calculus that do not occur in ordinary predicate calculus are given in Appendix A.

We will restrict our attention to generalization rules that transform one or more statements into a single more general statement:

$$\{D_i :: > K\}_{i \in I} \vdash D :: > K. \tag{17}$$

Such a rule states that if an event (a symbolic description of an object or situation) satifies any description D_i, $i \in I$, then it also satisfies description D (the reverse may not be true). A basic property of the generalization transformation is that the resulting description has 'unknown' truth-status, i.e., is a hypothesis that must be tested on new data. A generalization rule does not guarantee that the obtained description is useful or plausible.

We distinguish between two types of generalization rules: *selective* and *constructive*. If every descriptor used in the generated concept description D is among descriptors occurring in the initial concept descriptions D_i, $i = 1, 2, \ldots$, then the rule is selective, otherwise it is constructive.

5.2. Selective generalization rules

In the rules presented below, CTX, CTX_1 and CTX_2 stand for some arbitrary expressions (context descriptions) that are augmented by additional components to formulate a concept description.

- The *dropping condition rule* is a generalized version of the previously described rule (15)

$$\text{CTX} \,\&\, S :: > K \vdash \text{CTX} :: > K, \tag{18}$$

where S is an arbitrary predicate or logical expression.

This rule states that a concept description can be generalized by simply removing a conjunctively linked expression. This is one of the most commonly used rules for generalizing information.

- The *adding alternative rule*

$$\mathrm{CTX}_1 : :> K \models \mathrm{CTX}_1 \vee \mathrm{CTX}_2 : :> K. \tag{19}$$

A concept description can be generalized by adding, through the use of logical disjunction, an alternative to it. An especially useful form of this rule is when the alternative is added by extending the scope of permissible values of one specific descriptor. Such an operation can be expressed very simply by using the internal disjunction operator of the annotated predicate calculus. For example, suppose that a concept description is generalized by allowing objects to be not only red but also blue. This can be expressed as follows.

$$\mathrm{CTX} \,\&\, [\mathrm{color} = \mathrm{red}] : :> K \models \mathrm{CTX} \,\&\, [\mathrm{color} = \mathrm{red} \vee \mathrm{blue}] : :> K \tag{20}$$

(forms in brackets are selectors; the expressions on the right of ' = ' are called references (see Appendix A)).

Because of the importance of this special case, it will be presented as a separate general rule.

- The *extending reference rule*

$$\mathrm{CTX} \,\&\, [L = R_1] : :> K \models \mathrm{CTX} \,\&\, [L = R_2] : :> K, \tag{21}$$

where $R_1 \subseteq R_2 \subseteq \mathrm{DOM}(L)$ and $\mathrm{DOM}(L)$ denotes the domain of L.

In this rule, L is a term, and R_1 and R_2 (references) are internal disjunctions of values of L. References R_1 and R_2 can be interpreted as sets of values that descriptor L can take in order to satisfy the concept description.

The rule states that a concept description can be generalized by enlarging the reference of a descriptor ($R_2 \supseteq R_1$). The elements added to R_2 must, however, be from the domain of L.

If R_2 is extended to be the whole domain, i.e., $R_2 = \mathrm{DOM}(L)$, then the selector $[L = \mathrm{DOM}(L)]$ is always true, and therefore can be removed. In this case, the extending reference rule becomes the dropping condition rule. They take into consideration the type of the descriptor L (defined by the structure of $\mathrm{DOM}(L)$). They are presented as separate rules below.

- The *closing interval rule*

$$\left.\begin{array}{l} \mathrm{CTX} \,\&\, [L = a] : :> K \\ \mathrm{CTX} \,\&\, [L = b] : :> K \end{array}\right| \models \mathrm{CTX} \,\&\, [L = a \,..\, b] : :> K, \tag{22}$$

where L is a linear descriptor, and a and b are some specific values of descriptor L. The two premises are assumed to be connected by the logical conjunction (this convention holds for the remaining rules, as well).

The rule states that if two descriptions of the same class (the premises of the

rule) differ in the values of only one linear descriptor, then the descriptions can be replaced by a single description in which the reference of the descriptor is the interval linking these two values.

To illustrate this rule, consider as objects two states of a machine, and K as a class of *normal* states. The rule says that if a machine is in the normal state for two different temperatures, say, a and b, then a hypothesis is made that all states in which the temperature falls into the interval $[a, b]$ are also normal. Thus, this rule is not only a logically valid generalization rule, but expresses also some aspect of plausibility.

- The *climbing generalization tree rule*

$$\left.\begin{array}{l} \text{CTX} \,\&\, [L = a] : :> K \\ \text{CTX} \,\&\, [L = b] : :> K \\ \quad\vdots \\ \text{CTX} \,\&\, [L = i] : :> K \end{array}\right| \text{(one or more statements)} \;\prec\; \text{CTX} \,\&\, [L = s] : :> K, \tag{23}$$

where L is a structured descriptor, and s represents the lowest parent node whose descendants include nodes a, b, ... and i in the generalization tree domain of L.

The rule is applicable only to descriptions involving structured descriptors, and is used in various forms by, e.g., Winston [88], Hedrick [30], Lenat [42], Michalski [49], Michalski, Stepp and Diday [54], Mitchell [56, 57]. The following example illustrates the rule

$$\left.\begin{array}{l} \exists P, \text{CTX} \,\&\, [\text{shape}(P)=\text{triangle}] : :> K \\ \exists P, \text{CTX} \,\&\, [\text{shape}(P) = \text{pentagon}] : :> K \end{array}\right| \;\prec\; \begin{array}{l} \exists P, \text{CTX} \,\&\, \\ [\text{shape}(P)=\text{polygon}] : :> K. \end{array}$$

Paraphrasing this rule in English: if an object of class K is triangular and another object of this class is pentagonal, then the rule generates a statement that objects of class K are polygonal.

- The *turning constants into variables rule* is best known for the case of descriptive generalization

$$\left.\begin{array}{l} F[\text{a}] \\ F[\text{b}] \\ \quad\vdots \\ F[\text{i}] \end{array}\right| \text{(one or more statements)} \;\prec\; \forall v, F[v], \tag{24}$$

where $F[v]$ stands for some description (formula) dependent on variable v, and a, b, ... are constants.

some description $F[v]$ holds for v's being a constant a or constant b, etc., then the rule generalizes these observations into a statement that $F[v]$ holds for every value of v. This is the most often used rule in methods of inductive inference employing predicate calculus.

A corresponding rule for concept acquisition is

$$F[\mathrm{a}] \,\&\, F[\mathrm{b}] \,\&\, \cdots :: > K \models \exists v, F[v] :: > K. \tag{25}$$

To illustrate this version, assume that a, b, etc. are parts of an object of class K that have a property F. Rule (25) generalizes these facts into an assertion that if any part of an object has property F, then the object belongs to class K.

- The *turning conjunction into disjunction rule*

$$F_1 \,\&\, F_2 :: > K \models F_1 \vee F_2 :: > K, \tag{26}$$

where F_1 and F_2 are arbitrary descriptions.

A concept description can be generalized by replacing the conjunction operator by the disjunction operator.

- The *extending the quantification domain rule*, in the simplest case, changes the universal quantifier into the existential quantifier

$$\forall v, F[x] :: > K \models \exists v, F[v] :: > K. \tag{27}$$

This rule can be viewed as a generalization of the previous rule (26). Using the concept of numerical quantifier (see Appendix A) this rule can be expressed in an even more general way:

$$\exists(I_1)v, F[v] :: > K \models \exists(I_2)v, F[v] :: > K, \tag{28}$$

where I_1, I_2 are the quantification domains (sets of integers) satisfying relation $I_1 \subseteq I_2$.

For example, the statement 'if an object has two parts ($I_1 = \{2\}$) with property F, then it belongs to class K' can be generalized by rule (28) to a statement 'if an object has two or more parts ($I_2 = \{2, 3, \ldots\}$) with property F, then it belongs to class K'.

- The *inductive resolution rule*:

(a) As applied to concept acquisition. The deductive inference rule, called the resolution principle, widely used in automatic theorem proving, can be adopted as a rule of generalization for concept acquisition. In propositional form, the resolution principle can be expressed as

$$(P \Rightarrow F_1) \,\&\, (\sim P \Rightarrow F_2) \models F_1 \vee F_2, \tag{29}$$

where P is a predicate and F_1 and F_2 are arbitrary formulas. By interpreting both sides of (29) as concept descriptions, and making appropriate transformations, we obtain

$$\left.\begin{array}{l} P \,\&\, F_1 :: > K \\ \sim P \,\&\, F_2 :: > K \end{array}\right| \!\!< F_1 \vee F_2 :: > K. \tag{30}$$

To illustrate this rule, assume that K is the set of situations when John goes to a movie. Suppose that it has been observed that he goes to a movie when he has company (P) and the movie has high rating (F_1), or when he does not have company ($\sim P$), but has plenty of time (F_2). Rule (30) generalizes these two observations to a statement 'John goes to a movie either when the movie has high rating or he has plenty of time'.

(b) As applied to descriptive generalization. By applying the logical equivalence $(Q \models P) \Leftrightarrow (\sim P \models \sim Q)$ (the law of contraposition) to expression (29), then reversing the obtained rule and substituting the negative literals by the positive, we obtain

$$P \,\&\, F_1 \vee \sim P \,\&\, F_2 \mid\!\!< F_1 \,\&\, F_2. \tag{31}$$

This version has been formulated by Morgan [59].

Both versions, (a) and (b), can be generalized by applying the full-fledged resolution principle that uses predicates with arguments, and the unification algorithm to unify these arguments (e.g., [14]).

- The *extension against rule*

$$\left.\begin{array}{l} \mathrm{CTX}_1 \,\&\, [L = R_1] :: > K \\ \mathrm{CTX}_2 \,\&\, [L = R_2] :: > \sim K \end{array}\right| \!\!< [L \neq R_2] :: > K, \tag{32}$$

where sets R_1 and R_2 are assumed to be disjoint.

Given a description of an object belonging to class K (a positive example), and a description of an object not belonging to this class (a negative example), the rule produces the most general statement consistent with these two descriptions. It is an assertion that classifies an object as belonging to class K if descriptor L does not take any value from the set R_2, thus ignoring context descriptions CTX_1 and CTX_2. This rule is the basic rule for learning discriminant descriptions from examples used in the previously mentioned inductive program AQ11 [52]. Various modifications of this rule can be obtained by replacing reference R_2 in the output assertion by some superset of it (that does not intersect with R_1).

5.3. Constructive generalization rules

Constructive generalization rules generate inductive assertions that use descriptors not present in the original observational statements. This means that

the rules perform a transformation of the original representation space. The following is a general constructive rule that makes such a transformation by applying the knowledge of a relationship between different concepts. It is assumed that this relationship is known to the learning system as background knowledge, as a previously learned concept, or that it is computed according to user-defined procedures.

$$\left.\begin{array}{l} \mathrm{CTX} \,\&\, F_1 : :> K \\ F_1 \Rightarrow F_2 \end{array}\right| \prec \mathrm{CTX} \,\&\, F_2 : :> K \tag{33}$$

The rule states that if a concept description contains a part F_1 (a concept, a subdescription, etc.) that is known to imply some other concept F_2, then a more general description is obtained by replacing F_1 by F_2. For example, suppose a learning system is told that if an object is black, wide, and long, then it belongs to class K (e.g., is a blackboard). This can be expressed in the annotated predicate calculus:

$$\exists P, [\mathrm{color}(P) = \mathrm{black}][\mathrm{width}(P) \,\&\, \mathrm{length}(P) = \mathrm{large}] : :> K .$$

Suppose the learner already knows that

$$\forall P, ([\mathrm{width}(p) \,\&\, \mathrm{length}(P) = \mathrm{large}] \Rightarrow [\mathrm{area}(P) = \mathrm{large}]) .$$

Then rule (33) produces a generalization

$$\exists P, [\mathrm{color}(P) = \mathrm{black}][\mathrm{area}(P) = \mathrm{large}] : :> K .$$

As another example, suppose the system is given a description of an object classified as an arch. This description states that a horizontal bar is on top of two equal objects placed apart, B_1 and B_2, having certain color, weight, shape, etc. Suppose now that characterizations of B_1 and B_2 in this description satisfy a previously learned concept of a block. Then rule (33) generates an assertion that an arch is a bar on top of two placed-apart blocks. This rule is the basis for an interactive concept learning system developed by Sammut [72].

Specific constructive generalization rules can be obtained from (55) by evoking procedures computing new descriptors in expression F_2 as functions of initial or previously derived descriptors (contained in F_1). Here are some examples of rules for generating new descriptors.

- *Counting arguments rules.*

(a) The CQ rule (count quantified variables). If a concept description is in the form

$$\exists v_1, v_2, \ldots, v_k, F[v_1, v_2, \ldots, v_k] ,$$

then the rule generates descriptors '#v_COND' representing the number of v_i's that satisfy some condition COND. This condition expresses selected properties of v_i's specified in the concept description. Since many such CONDs can usually be formulated, the rule allows the system to generate a large number of such descriptors.

For example, if the COND is '[attribute$_1$(v_i) = R]', then the generated descriptor will be '#$_i$_attribute$_1$_R' counting the number of v_i's that satisfy this condition. If the attribute$_1$ is, for instance, length, and R is [2..4], then the derived descriptor is '#v_i_length_2..4' (i.e., it measures the number of v_i's whose length is between 2 and 4, inclusively).

(b) The CA-rule (count arguments of a predicate). If a descriptor in a description is a relation with several arguments, REL(v_1, v_2, ...), the rule generates descriptors '#v_COND', measuring the number of arguments in REL that satisfy some condition COND. Similar to the above, many such descriptors can be generated, each with different COND.

The annotation of a descriptor provides information about its properties. Such a property may be that a descriptor is, for example, a transitive relation, such as relations 'above', 'inside', 'left-of', and 'before'. For example, if the relation is 'contains(A, B_1, B_2, ...)', stating that object A contains objects B_1, B_2, ..., and COND is 'large and red', then the derived descriptor '#B_large_red_A_contains' measures the number of B_i's contained in A that are large and red.

- The *generating chain properties rule.* If the arguments of different occurrences of a transitive relation in a concept description form a chain, that is, form a sequence of consecutive objects ordered by this relation, the rule generates descriptors characterizing some specific objects in the chain. Such objects may be

LST-object—the 'least object', i.e., the object at the beginning of the chain (e.g., the bottom object in the case of the relation 'above'),

MST-object—the object at the end of the chain (e.g., the top object),

MID-object—the objects in the middle of the chain,

Nth-object—the object at the Nth position in the chain (starting from LST-object).

After identifying these objects, the rule investigates all known properties of them (as specified in the observational statements) in order to determine potentially relevant new descriptors. The rule also generates a descriptor characterizing the chain itself, namely, REL-chain-length—the length of the chain defined by relation REL.

For example, if the REL is ON-TOP, then descriptor ON-TOP-chain-length would specify the height of a stack of objects. When a new description is generated and adopted, an annotation for it is also generated and filled out, as in Lenat [42]. This rule can be extended to a partial order relation. In such a case it becomes the 'find extrema of a partial order' rule.

The *detecting descriptor interdependence rule.* Suppose that given is a set of objects exemplifying some concept, and that attribute descriptions are used to characterize these objects. Such descriptions specify only attribute values of the objects; they do not characterize the objects' structure. Suppose that the values a linear descriptor x takes on in all descriptions (events) are ordered in increasing order. If the corresponding values of another linear descriptor y exhibit an increasing or decreasing order, then a two-place descriptor $M(x, y)$ is created, signifying that x and y have a *monotonic relationship.* This descriptor has value $\uparrow$ when y values are increasing and value $\downarrow$ when they are decreasing.

The idea of the above M-descriptor can be extended in two directions. The first is to create M-descriptors dependent on some condition COND that must be satisfied by the events under consideration:

$$M(x, y)\_\text{COND}\,.$$

For example, descriptor

$$M(\text{length, weight})\_\text{red}$$

states that length and weight have a monotonic relationship for red objects.

The second direction of extension is to relax the requirement for the monotonic relationship, i.e., not to require that the order of y values is strictly increasing (or decreasing), but only approximately increasing (or decreasing). For example, the coefficient of statistical correlation between x and y can be measured, and when its absolute value is above a certain threshold, a descriptor $R(x, y)$ is created. The domain of this R-descriptor can also be $\{\uparrow, \downarrow\}$, indicating the positive or negative correlation, respectively, or it can have values representing several subranges of the correlation coefficient. Similarly, as in the case of M-descriptors, R-descriptors can be extended to R-COND descriptors.

The M- or R-descriptors can be used to generate new descriptors. For example, if $[M(x, y) = \uparrow]$, then a new descriptor $z = x/y$ can be generated. If z assumes a constant or a nearly constant value, then an important relationship has been discovered. Similarly, if $[M(x, y) = \downarrow]$, then a new descriptor $z = x \cdot y$ can be generated. These two techniques for generating new descriptors have been successfully used in the BACON system for discovering mathematical expressions representing physical or chemical laws [39].

The above ideas can be extended to structural descriptions. Such descriptions involve not only global properties of objects, but also properties of objects' parts and the relationships among the parts. Suppose that in a structural description of an object, existentially quantified variables $P_1, P_2, \ldots, P_m$ denote its parts. If $x(P_i)$ and $y(P_i)$ and $y(P_i)$ are linear descriptors of P_i (e.g.,

numerical attributes characterizing parts P_i, $i = 1, 2, \ldots$), the above described techniques for generating M- and R-descriptors can be applied.

6. The STAR Methodology

6.1. The concept of a star

The methodology presented here for learning structural descriptions from examples receives its name from the major concept employed in it, that of a *star*. In the most general sense, a *star of an event e* (a description of a single object or situation) *under constraints F*, is a set of all possible alternative nonredundant descriptions of event e that do not violate constraints F. A somewhat more restrictive definition of a star will be used here. Let e be an example of a concept to be learned and F be a set of some counterexamples of this concept. A star of the *event e against* the event set F, denoted $G(e|F)$, is defined as the set of all maximally general c-expressions that cover (i.e., are satisfied by) event e and that do not cover any of the negative events in F.

The c-expressions in a star may contain *derived* descriptors, i.e., descriptors not present in the observational statements. In such a case, testing whether event e satisfies a given description requires that appropriate transformations be applied to the event. Such a process can be viewed as proving that the event implies the description, and therefore methods of automatic theorem proving could be used.

In practical problems, a star of an event may contain a very large number of descriptions. Consequently, such a theoretical star is replaced by a *reduced star* $RG(e|F, m)$ that contains no more than a fixed number, m, of descriptions. These m descriptions are selected as the m most preferable descriptions among the remaining ones according to the preference criterion defined in the problem background knowledge. Variable m is a parameter of the learning program, defined either by the user or by the program itself, as a function of the variable computational resources.

Papers [50, 54] give an illustration and an algorithm for generating a reduced star with c-expressions restricted to attribute expressions (i.e., expressions involving only object attributes). Section 6.3 presents an algorithm for generating a reduced star consisting of regular c-expressions. The concept of a star is useful because it reduces the problem of finding a complete description of a concept to subproblems of finding consistent descriptions of single positive examples of the concept.

Since any single example of a concept can always be characterized by a conjunctive expression (a logical product of some predicates), elements of a star can always be represented by conjunctive descriptions. One should also notice that if the concept to be learned is describable by a c-expression, then this description clearly will be among the elements of a (nonreduced) star of *any*

single positive example of the concept. Consequently, if there exists a positive example not covered by any description of such a star, then the complete concept description must be disjunctive, i.e., must include more than one c-expression.

6.2. Outline of the general algorithm

It is assumed that every observational statement is in the form

$$\text{a-expression} : :> K, \tag{34}$$

where a-expression is an atomic expression describing an object (recall Section 4.6) and K is the concept exemplified by this object.

It is also assumed that inductive assertions are in the form of a single c-expression or the disjunction of c-expressions. For simplicity we will restrict our attention to only single-concept learning. In the case of multiple-concept learning, the algorithm is repeated for each concept with modifications depending on the assumed interdependence among the concept descriptions (Section 2.3).

Let POS and NEG denote sets of events representing positive and negative examples of a concept, respectively. A general and simplified version of the STAR algorithm can be described as follows.

Step 1. Select randomly an event e from POS.

Step 2. Generate a reduced star, $RG(e|\text{NEG}, m)$, of the event e against the set of negative examples NEG, with no more than m elements. In the process of star generation apply generalization rules (both selective and constructive), task-specific rules, and heuristics for generating new descriptors supplied by problem background knowledge, and definitions of previously learned concepts.

Step 3. In the obtained star, find a description D with the highest preference according to the assumed preference criterion LEF.

Step 4. If description D covers set POS completely, then go to Step 6.

Step 5. Otherwise, reduce the set POS to contain only events not covered by D, and repeat the whole process from Step 1.

Step 6. The disjunction of all generated descriptions D is a complete and consistent concept description. As a final step, apply various reformulation rules (defined in the problem background knowledge) and 'contracting' rules ((A.8) and (A.9)) in order to obtain a possibly simpler expression.

This algorithm is a simplified version of the general covering algorithm A^q [47, 48]. The main difference is that algorithm A^q selects the initial events (if possible) from events not covered by any of the descriptions of generated stars, rather than not covered by only the selected descriptions D. This way the algorithm is able to determine a bound on the maximum number of separate

descriptions in a disjunction needed to define the concept. Such a process may, however, be computationally very costly.

The above algorithm describes only single-step learning. If after generating a concept description, a newly presented training event contradicts it, specialization or generalization rules are applied to generate a new, consistent concept description. A method for such incremental learning is described by Michalski and Larson [52].

The central step in the above algorithm is the generation of a reduced star. This can be done using a variety of methods. Thus, the above STAR algorithm can be viewed as a general schema for implementing various learning methods and strategies. The next section describes one specific method of star generation.

6.3. Star generation: the INDUCE method

This method generates a reduced star $RG(e|\text{NEG}, m)$ by starting with a set of single selectors, which are either extracted from the event for which the star is generated or inferred from the event by applying constructive generalization rules or inference rules provided by background knowledge. These selectors are then specialized by adding other selectors until consistency is achieved (i.e., until each expression does not intersect with set NEG). Next, the obtained consistent expressions are generalized so that each achieves the maximum coverage of the remaining positive training examples. The best m so-obtained consistent and generalized c-expressions (if some are also complete, then they are alternative solutions) constitute the sought reduced star $RG(e|\text{NEG}, m)$. Specifically, the steps of the procedure are as follows.

(1) In the first step individual selectors of event e are put on the list called PS. This list is called a *partial star*, because its elements may cover some events in NEG. These initial elements of PS (single selectors from e) can be viewed as generalizations of event e obtained by applying in all possible ways the dropping condition generalization rule (each application drops all selectors except one). Elements of the partial star PS are then ordered from the most to the least preferred according to a preference criterion

$$\text{LEF}_1 = \langle(-\text{negcov}, \tau_1), (\text{poscov}, \tau_2)\rangle\,, \tag{35}$$

where negcov and poscov are numbers of positive and negative examples, respectively, covered by an expression in the star, and τ_1, τ_2 are tolerances (recall Section 4.7). The LEF_1 minimizes the negcov (by maximizing the $-$negcov) and maximizes poscov.

(2) The list PS is then expanded by adding new selectors obtained by applying the following inference rules to the event e:

(a) the constructive generalization rules (Section 5.3),

(b) the problem-specific heuristics defined in the background knowledge,

(c) the definitions of the previously learned concepts (to determine whether parts of *e* satisfy some already known concepts).

(3) Each new selector is inserted in the appropriate place in list PS, according to preference criterion LEF_1. The size of PS is kept within the limit defined by parameter *m* by removing from PS all but the *m* most preferred selectors.

(4) Descriptions in PS are tested for consistency and completeness. A description is consistent if negcov = 0 (i.e., if it covers no events in NEG), and is complete if poscov is equal to the total number of positive examples. Consistent and complete descriptions are removed from PS and put on the list called SOLUTIONS. If the size of the list SOLUTIONS is greater than a parameter #SOL, then the algorithm stops. Parameter #SOL determines the number of desired alternative concept descriptions. Incomplete but consistent descriptions are removed from the list PS and put on the list called CONSISTENT. If the size of the CONSISTENT list is greater than a parameter #CONS, then control is transferred to Step 6.

(5) Each expression in PS is specialized in various ways by appending to it a single selector from the original list PS. Appended selectors must be of lower preference than the last selector in the conjunctive expression (initially, the expression has only one selector). Parameter %BRANCH specifies the percentage of the selectors ranked lower (by the preference criterion) than the last selector in the current conjunction. If %BRANCH = 100%, all lower preference selectors are singly appended—that is, the number of new expressions generated from this conjunction will be equal to the total number of selectors having lower preference than the last selector in the conjunction. All new obtained expressions are ranked by LEF_1 and only the *m* best are retained. This 'expression growing' process is illustrated in Fig. 1.

Steps 4 and 5 are repeated until the CONSISTENT list contains the number of expressions specified by parameter #CONS, or until the time allocated for this process is exhausted.

(6) Each expression on the CONSISTENT list is generalized by applying the extension against, closing the interval, and climbing generalization tree generalization rules. An efficient way to implement such a process is to transform the original structural description space into an attribute description space. Attributes (i.e., descriptions with zero arguments) defining this space are created from the descriptors in the given expression on the CONSISTENT list in a manner such as that described in [20]. The generalization of the obtained attribute descriptions is accomplished by the attribute star generation procedure, analogous to the one described by Michalski, Stepp and Diday [54]. Details of this process of transforming structural descriptions into attribute descriptions are described by Larson [40]. The reason for such a transformation is that structural descriptions are represented as labeled graphs, while attribute descriptions are represented as binary strings. It is computationally much more economical to handle binary strings than labeled graphs.

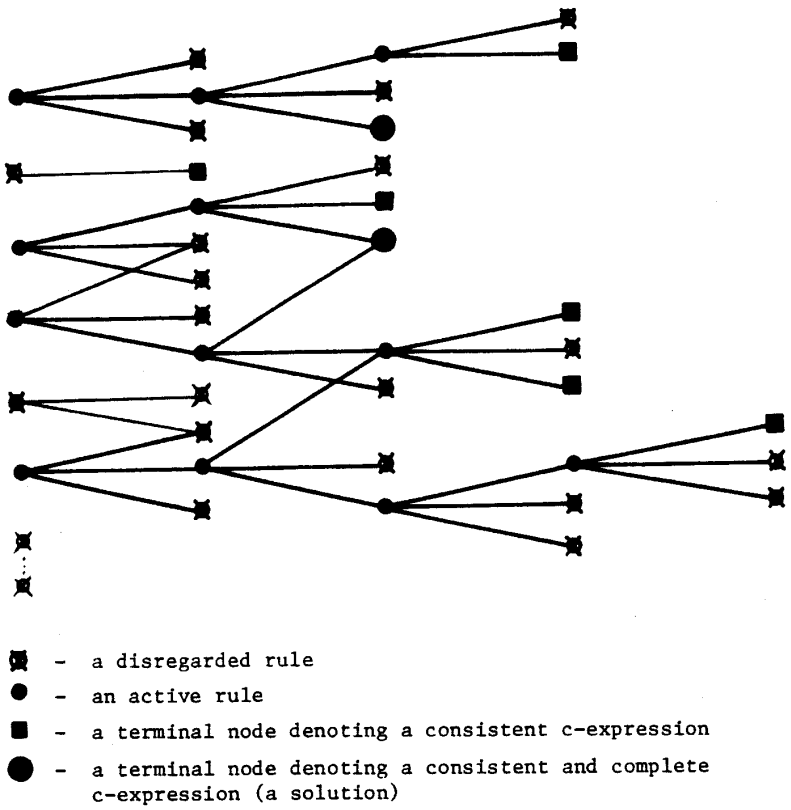

FIG. 1. Illustration of the process of generating a reduced star $RG(e/\text{NEG}, m)$. The nodes in the first column are selectors extracted from the event e or derived from e by applying inference rules. Each arc represents an operation of adding a new selector to the current c-expression.

(7) The obtained generalizations are ranked according to the global preference criterion LEF defined in the background knowledge. A typical LEF is to maximize the number of events covered in POS set and to minimize the complexity of the expression (measured, for example, by the number of selectors it contains). The m best expressions so determined constitute the reduced star $RG(e|\text{NEG}, m)$.

A somewhat restricted version of the above-described INDUCE method and STAR algorithm has been implemented in various versions of the INDUCE learning program [40, 18, 49, 32].

7. An Example

To illustrate the inductive learning methodology just presented, let us consider a simple problem in the area of conceptual data analysis. Suppose we are given examples of 'cancerous' and 'normal' cells, denoted DNC and DNN, respectively, in Fig. 2, and the task of the analysis is

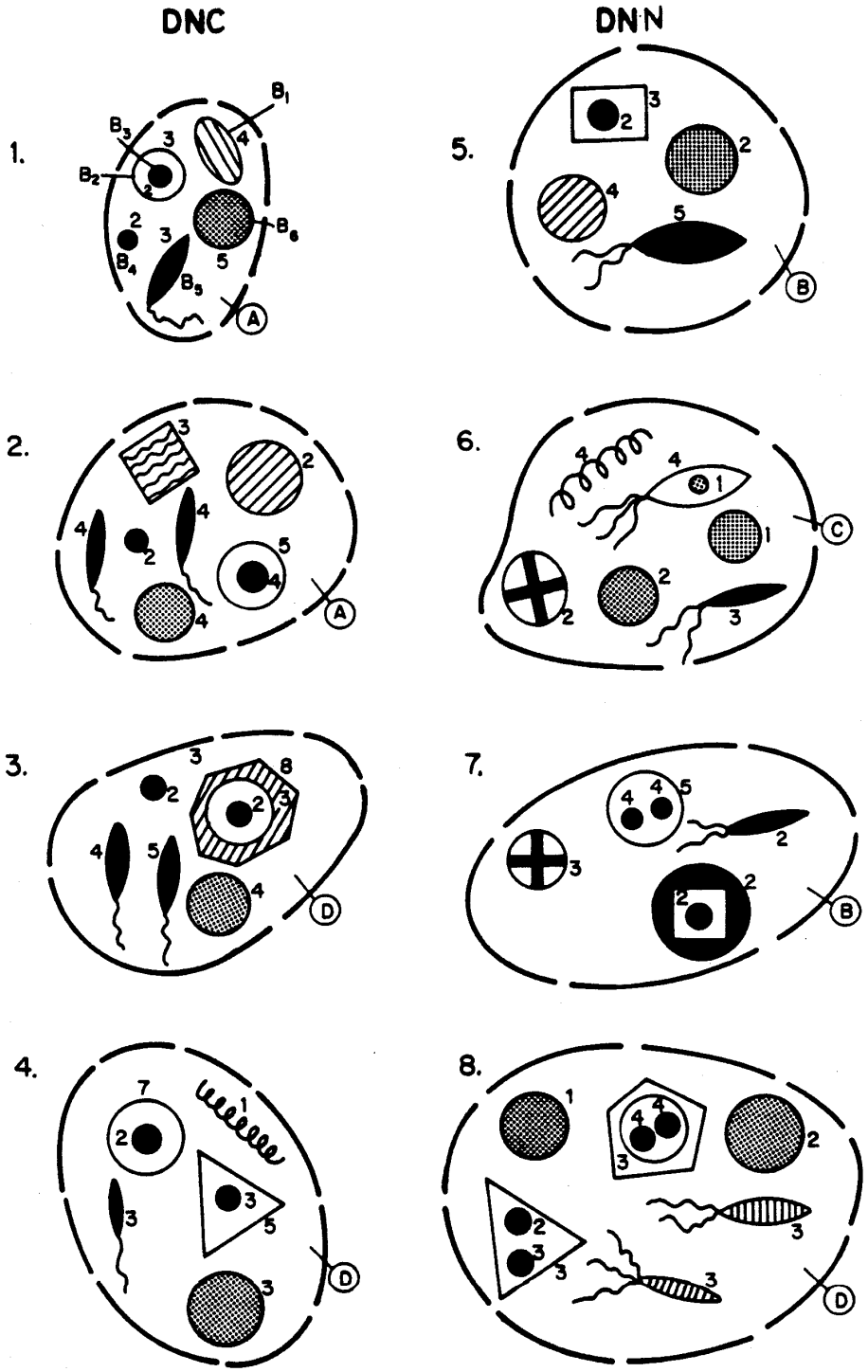

FIG. 2.

(1) to determine properties differentiating the two classes of cells (i.e., to find discriminant descriptions of each class),

(2) to determine important common properties of the cancerous and the normal cells (i.e., to find a characteristic description of each class).

An assumption is made that the properties to be discovered may involve both quantitative information about the cells and their components, and qualitative information, which includes nominal variables and relationships existing among the components.

The solution to the problem posed (or similar problems) can be obtained by a successive repetition of the 'focus attention → hypothesize → test' cycle described below.

The 'focus attention' phase is concerned with defining the scope of the problem under consideration. This includes selecting descriptors appearing to be relevant, specifying underlying assumptions, and formulating the relevant problem knowledge. This first phase is performed by a researcher; it involves his/her technical knowledge and informal intuitions. The third, the 'test' phase, examines the hypotheses and tests them on new data. This phase may require collecting new samples, performing laboratory experiments, and/or critically analyzing the hypotheses. This phase is likely to involve knowledge and abilities that go beyond currently feasible computer systems.

It is the second, the 'hypothesize' phase, in which an inductive learning system may play a useful role: the role of an assistant for conducting a search for the most plausible and/or most interesting hypotheses. This search may be a formidable combinatorial task for a researcher, if the data sample is large and if each item of the data (in this case, a cell) is described by many variables and/or relations.

Individual steps are as follows.

(1) The user determines the set of initial descriptors and provides an annotation for each descriptor. We will assume that the annotation specifies the type, the domain, and any special properties of each descriptor (e.g., the transitivity of a relation). In the case of structured descriptors, the annotation also specifies the structure of the domain. The specification of the annotation constitutes the first part of the problem background knowledge.

Suppose that for our simple example problem, the following descriptors are selected.

Global descriptors (descriptors characterizing a whole cell):

- circ: the number of segments in the circumference of the cell,
  type: linear,
  domain: {1 . . 10};

(II) - pplasm: the type of protoplasm in the cell (marked by encircled capital letters in Fig. 2),
  type: nominal,
  domain: {*A*, *B*, *C*, *D*};

(II) *Local descriptors* (those characterizing cell bodies and their relationships:

- shape(B_i): the shape of body B_i,
  type: structured,
  domain: a tree structure with a set of leaves {triangle, circle, ellipse, heptagon, square, boat, spring},
  nonleaf nodes are defined by rules:

  [shape = circle ∨ ellipse] ⇒ [shape = oval],
  [shape = triangle ∨ square ∨ heptagon] ⇒ [shape = polygon],
  [shape = oval ∨ polygon] ⇒ [shape = regular],
  [shape = spring ∨ boat] ⇒ [shape = irregular];

- texture(B_i): the texture of body B_i,
  type: nominal,
  domain: {blank, shaded, crossed, wavy, solid-black, solid-grey, stripes};
- weight(B_i): the weight of body B_i,
  type: linear,
  domain: {1, 2, . . . , 5};
- orient(B_i): the orientation of B_i,
  type: linear-cyclic (the last element is followed by the first),
  domain: {N, NE, E, SE, S, SW, W, NW},
  condition of applicability: if [shape(B_i) = boat];
- contains (C, B_1, B_2, . . .) – C contains B_1, B_2, . . .
  type: nominal,
  domain: {True, False},
  properties: transitive relation;
- hastails(B, L_1, L_2, . . .): a body B has tails L_1, L_2, . . .
  type: nominal,
  domain: {True, False},
  condition of applicability: if [shape(B) = boat].

Note that the descriptors 'contains' and 'hastails' are predicates with variable number of arguments. Descriptor 'contains' is characterized as a transitive relation. Descriptors 'hastails' and 'orient' are applicable only under a certain condition.

(2) The user formulates observational statements, which describe cells in terms of selected descriptors and specify the class to which each cell belongs. For example, the following is an observational statement for the DNC cell 1.

$$\exists\,.\,\mathrm{CELL}_1, B_1, B_2, \ldots, B_6\,[\mathrm{contains}(\mathrm{CELL}_1, B_1, B_2, \ldots, B_6)]$$
$$[\mathrm{circ}(\mathrm{CELL}_1) = 8]\,[\mathrm{pplasm}(\mathrm{CELL}_1) = A]\,[\mathrm{shape}(B_1) = \mathrm{ellipse}]\,\&$$
$$[\mathrm{texture}(B_1) = \mathrm{stripes}]\,[\mathrm{weight}(B_1) = 4]\,[\mathrm{orient}(B_1) = \mathrm{NW}]\,\&$$
$$[\mathrm{contains}(B_2, B_3)]\,[\mathrm{texture}(B_2) = \mathrm{blank}]\,[\mathrm{weight}(B_2) = 3]\cdots\,\&$$
$$[\mathrm{shape}(B_6) = \mathrm{circle}]\,[\mathrm{texture}(B_6) = \mathrm{shaded}]\,[\mathrm{weight}(B_6) = 5]$$
$$::>[\mathrm{class} = \mathrm{DNC}].$$

(3) To specify the second part of the problem background knowledge the

user indicates which general rules of constructive induction (Section 5.3) are applicable, and also formulates any problem-specific rules.

The constructive rules will generate various new derived descriptors. For example, the counting rule CQ will generate, among others, a descriptor:

- #B-black-boat: the number of bodies whose shape is 'boat' and texture is 'solid-black', i.e., assuming COND

$$[\text{shape}(B) = \text{boat}][\text{texture}(B) = \text{solid-black}].$$

(For simplicity of notation, the name of this descriptor, as well as other descriptors below, has been abbreviated, so it does not follow strictly the naming convention described in Section 5.3.) The counting rule CA will generate such descriptors as

- total-B: the total number of bodies in a cell (no COND is used);
- indep-B: the number of independent bodies in a cell, assuming the COND 'bodies not contained in another body';
- #contained-in-B: the number of smaller bodies contained in the body B;
- #tails-boat-B: the number of tails in a body B, whose shape is 'boat'.

As advice to the system, the user may formulate arbitrary arithmetic expressions for generating possibly relevant descriptors. For example, the user may suggest a descriptor:

$$\text{weight(CELL)} = \sum_i \text{weight}(B_i),$$

where B_i, $i = 1, 2, \ldots$ denote bodies in a cell.

The background knowledge may also contain special concepts—even or odd number, the definitions of the area and perimeter of a circle or rectangle, etc.

(4) Finally, as the last part of the background knowledge, the user specifies the type of description sought and the hypothesis preference criterion. Let us assume that both maximal characteristic descriptions and minimal discriminant descriptions are sought. We therefore choose as the preference criterion for constructing characteristic descriptions: 'maximize the length of generated complete c-expressions', and for constructing discriminant descriptions: 'minimize the length of consistent and complete c-expressions'.

For illustration, we shall present here samples of discriminant descriptions and selected components of a characteristic description of the DNC 'cells', obtained by the INDUCE program[2].

Discriminant descriptions of DNC cells. Each of these descriptions is sufficient to discriminate all DNC cells from DNN cells. A concept description for class DNC can thus be any one of these descriptions or the disjunction of two or more of these descriptions.

$$\exists(1)B\,[\text{texture}(B) = \text{shaded}][\text{weight}(B) \geq 3].$$

[2]It may be instructive to the reader to try at this point to formulate his/her own descriptions.

'Every DNC cell, as opposed to DNN, has exactly one body with 'shaded' texture and weight at least 3.' (Paraphrasing in English.)

$$\exists[\text{circ} = \text{even}] .$$

'The number of segments in the circumference of every DNC cell is even'. (The concept of 'even' was determined by 'climbing the generalization tree' rule.)

$$\exists(\geq 1)\text{B}\,[\text{shape}(B) = \text{boat}][\text{orient}(B) = \text{N} \vee \text{NE}] .$$

'Every DNC cell has at least one 'boat' shape body with orientation N or NE'.

$$\exists(\geq 1)\text{B}[\underline{\#\text{tails-boat-}B} = 1] .$$

'Every DNC cell has at least one body with number of tails equal to 1.'

$$\exists(1)B[\text{shape}(B) = \text{circle}][\underline{\#\text{contains-}B} = 1] .$$

'Every DNC cell has a circle containing a single object.' (A related and somewhat redundant description is that every cell contains a circle that has another solid black circle inside it.)

Underscored descriptors are derived descriptors obtained through constructive generalization rules.

Characteristic descriptions of DNC cells. Every description below is a characterization of some pattern common to all DNC cells. Some of these patterns taken separately may cover one or more DNN cells (unlike the discriminant descriptions). In contrast to discriminant descriptions, the length of each description has been maximized rather than minimized.

$$\forall(1)B\,[\text{weight}(B) = 5] .$$

'In every DNC cell there is one and only one body with weight 5.' (Paraphrasing in English.)

$$\begin{aligned}\exists.(2)B_1, B_2\,&[\text{contains}(B_1, B_2)][\text{shape}(B_1)\text{shape}(B_2) = \text{circle}]\\ &[\text{texture}(B_1) = \text{blank}][\text{weight}(B_1) = \text{odd}]\\ &[\text{texture}(B_2) = \text{solid\_black}][\text{weight}(B_2) = \text{even}]\\ &[\underline{\#\text{contained\_in\_}B_1} = 1] .\end{aligned}$$

'In every cell there are two bodies of circle shape, one contained in another, of which the outside circle is blank, and has 'odd' weight, the inside circle is solid black and has 'even' weight. The number of bodies in the outside circle is only one'. (This is also a discriminant description but is not minimal.)

$$\exists(1)B\ [\text{shape}(B) = \text{circle}][\text{texture}(B) = \text{shaded}][\text{weight}(B) \geqslant 3]\ .$$

'Every cell contains a circle with 'shaded' texture, whose weight is at least 3'. (This is also a non-minimal discriminant description.)

$$\exists(>1)B\ [\text{shape}(B) = \text{boat}][\text{orient}(B) = \text{N} \vee \text{NE}][\underline{\#\text{tails-boat}(B)} = 1].$$

'Every cell has at least one body of 'boat' shape with N or NE orientation, which has one tail.' (This is also a non-minimal discriminant description.)

$$\exists(2)B\ [\text{shape}(B) = \text{circle}][\text{texture}(B) = \text{solid\_black}]$$

or alternatively

$$[\underline{\#B\_\text{circle\_solid\_black}} = 2]\ .$$

'Each cell has exactly two bodies that are solid black circles.'

$$[\text{pplasm} = A \vee D]$$

'The protoplasm of every cell is of type A or D.'

The above example is too simple for really unexpected patterns to be discovered. But it illustrates well the potential of the learning program as a tool for searching for patterns in complex data, especially when the relevant properties involve both numerical and structural information about the objects under consideration. An application of this program to a more complex problem [49] did generate unexpected patterns.

8. Conclusion

A theory of inductive learning has been presented that views such learning as a heuristic search through a space of symbolic descriptions, generated by an application of certain inference rules to the initial observational statements (teacher-generated examples of some concepts or environment-provided facts). The process of generating the goal description—the most preferred inductive assertion—relies on the universally intertwined and complementary operations of specializing or generalizing the currently held assertion in order to accommodate new facts. The domain background knowledge has been shown to be a necessary component of inductive learning, which provides constraints, guidance and a criterion for selecting the most preferred inductive assertion.

Such characterization of inductive learning is conceptually simple, and constitutes a theoretical framework for describing and comparing learning methods, as well as developing new methods. The STAR methodology for

learning structural descriptions from examples, described in the second part of the chapter, represents a general approach to concept acquisition which can be implemented in a variety of ways and applied to different problem domains.

There are many important topics of inductive learning that have not been covered here. Among them are learning from incomplete or uncertain information, multistage learning, learning from descriptions containing errors, learning with a multitude of forms of given observational statements, as well as multimodel-based inductive assertions, and learning general rules with exceptions. The problem of discovering new concepts, descriptors and, generally, various many-level transformations of the initial description space (the problem of constructive inductive learning) has been covered only very superficially.

These and related topics have been given little attention so far in the field of machine learning. There is no doubt, however, that as the understanding of the fundamental problems in the field matures, these challenging topics will be given increasing attention.

Appendix A. Annotated Predicate Calculus (APC)

This appendix presents definitions of the basic components of the annotated predicate calculus and some rules for equivalence-preserving transformations of APC expressions (rules that are nonexistent in the ordinary calculus) follow.

A.1. Elementary and compound terms

Terms can be elementary or compound. An *elementary term* (an e-term) is the same as a term in predicate calculus, i.e., a constant, a variable, or a function symbol followed by a list of arguments that are e-terms. A *compound term* (c-term) is a *composite* of elementary terms or is an e-term in which one or more arguments are such composites. The composite of e-terms is defined as the *internal conjunction* (&) or *internal disjunction* ($\vee$) of e-terms. (The meaning of these operators is explained later.) The following are examples of compound terms:

$$\text{RED} \vee \text{BLUE} \tag{A.1}$$

$$\text{height}(\text{BOX}_1 \,\&\, \text{BOX}_2)\,, \tag{A.2}$$

where RED, BLUE, BOX_1, BOX_2 are constants. Expression (A.1) and the form in parentheses in (A.2) are composites. Note that expressions (A.1) and (A.2) are not logical expressions that have a truth status (i.e., that can be true or false); they are to be used only as arguments of predicates. A compound term in which arguments are composites can be transformed (expanded) into a composite of elementary terms. Let f be an n-argument function whose $n-1$ arguments are represented by list A, and let t_1 and t_2 be elementary terms. The rules for

performing such a transformation, expressed as term-rewriting rules, are

$$f(t_1 \vee t_2, A) \leftrightarrow f(t_1, A) \vee f(t_2, A), \tag{A.3}$$

$$f(t_1 \,\&\, t_2, A) \leftrightarrow f(t_1, A) \,\&\, f(t_2, A). \tag{A.4}$$

If list A itself contains composites, then it is assumed that the internal disjunction is expanded first, followed by the internal conjunction (i.e., the conjunction binds stronger than the disjunction). Thus, term (A.2) can be transformed into a composite

$$\text{height}(\text{BOX}_1) \,\&\, \text{height}(\text{BOX}_2). \tag{A.5}$$

A.2. Elementary and compound predicates

Predicates also can be elementary or compound. An *elementary predicate* is the same as a predicate in the predicate calculus, i.e., a predicate symbol followed by a list of arguments that are e-terms. In a *compound predicate* one or more arguments is a compound term. For example, the following are compound predicates

$$\text{Went}(\text{Mary} \,\&\, \text{Mother}(\text{Stan}), \text{Movie} \vee \text{Theatre}), \tag{A.6}$$

$$\text{Inside}(\text{Key}, \text{Drawer}(\text{Desk}_1 \vee \text{Desk}_2)). \tag{A.7}$$

The meaning of a compound predicate is defined by rules for transforming it into an expression made of elementary predicates and ordinary 'external' logic operators of conjunction (&) and disjunction ($\vee$). We denote the internal and external operators identically, because they can be easily distinguished by the context (note that there is no distinction between them in natural language). If an operator connects predicates, then it is an external operator; if it connects terms, then it is an internal operator.

Let t_1 and t_2 be e-terms and P an n-ary predicate whose last $n-1$ arguments are represented by a list A. We have the following reformulation rules (i.e., equivalence preserving transformations of descriptions)

$$P(t_1 \vee t_2, A) \models P(t_1, A) \vee P(t_2, A), \tag{A.8}$$

$$P(t_1 \,\&\, t_2, A) \models P(t_1, A) \,\&\, P(t_2, A). \tag{A.9}$$

If an argument of a predicate is a compound term that is not a composite of elementary terms, then it is transformed first into a composite by rules (A.3) and (A.4). If A contains a composite of terms, then the disjunction is expanded first before conjunction (similarly as in expanding compound terms).

Rules (A.3), (A.4), (A.8) and (A.9) can be used as bidirectional transformation rules. By applying them forward (from left to right), a compound predicate can be expanded into an expression containing only elementary predicates, and by applying them backward, an expression with elementary predicates can be contracted into a compound predicate.

For example, by applying forward rule (A.8) and then (A.9), one can expand the compound predicate (A.6) into

$$\begin{aligned}&\text{Went(Mary, movie)} \,\&\, \text{Went(Mother(Stan), movie)} \vee \\ &\text{Went(Mary, theatre)} \,\&\, \text{Went(Mother(Stan), theatre)}\,. \end{aligned} \qquad \text{(A.10)}$$

Comparing logically equivalent expressions (A.6) and (A.10), one can notice that (A.6) is considerably shorter than (A.10), and in contrast to (A.10), represents explicitly the fact that Mary & Mother(Stan) went to the same place. Also, the structure of (A.6) is more similar to the structure of the corresponding natural language expression.

A.3. Relational statements

A simple and often used way of describing objects or situations is to state the values of selected attributes applied to these objects or situations. Although such information can be represented by predicates, this is not the most readable or natural way. The APC uses for this purpose a statement

$$\text{eterm}_i = a\,, \qquad \text{(A.11)}$$

stating that e-term$_i$ evaluates to a constant a. Such a statement is called an *atomic relational statement* (or an *atomic selector*). Expression (A.11) is a special case of a *relational statement* (also called *selector*), defined as

$$\text{Term}_1 \text{ rel } \text{Term}_2\,, \qquad \text{(A.12)}$$

where Term$_1$ and Term$_2$ are elementary or compound terms, and rel stands for one of the relational symbols: $=, \geqslant, >, <, \leqslant$.

If Term$_1$ and Term$_2$ are both elementary, then (A.12) states that the value of the function represented by Term$_1$ is in relation rel to the value of function represented by Term$_2$. For example, the expression

$$\text{distance(Boston, Tampa)} = \text{distance(Washington, Dallas)} \qquad \text{(A.13)}$$

states that the distance between Boston and Tampa is the same as the distance between Washington and Dallas. If Term$_2$ is a constant, then it evaluates to itself.

Expression (A.12) can be represented by a predicate

$$\text{rel}(\text{Term}_1, \text{Term}_2)\,. \tag{A.14}$$

If Term_1 or Term_2 is compound, or if both are, then the meaning of (A.12) is defined by expanding it into a form containing only relational statements with elementary terms. The expansion is performed by transforming (A.12) into (A.14), applying transformation rules (A.3), (A.4), (A.8) and (A.9), and then converting the elementary predicates into relational statements.

For example, a relational statement

$$\text{color}(P_1 \vee P_2) = \text{Red} \vee \text{Blue} \tag{A.15}$$

can be expanded into an expression

$$(\text{color}(P_1) = \text{Red} \vee \text{Blue}) \vee (\text{color}(P_2) = \text{Red} \vee \text{Blue}) \tag{A.16}$$

and finally to an expression consisting of only atomic selectors:

$$\begin{aligned}&(\text{color}(P_1) = \text{Red} \vee \text{color}(P_1) = \text{Blue}) \vee \\ &(\text{color}(P_2) = \text{Red} \vee \text{color}(P_2) = \text{Blue})\,.\end{aligned} \tag{A.17}$$

The two selectors in the disjunction (A.16) are examples of a *referential selector*, defined as a form

$$\text{Term}_1 \,\text{rel}\, \text{Term}_2\,, \tag{A.18}$$

where Term_1 (called *referee*) is a nonconstant elementary term and Term_2 (called *reference*) is a constant or the disjunction of constants from the domain of Term_1. If relation rel is '=' and Term_2 is the disjunction of some constants, then the referential selector (A.18) states that the function represented by Term_1 evaluates to one of the constants in Term_2. The referential selector is very useful for representing concept descriptions.

If the reference of a referential selector contains a sequence of consecutive constants from the domain of a linear descriptor, then the range operator '. .' is used to simplify the expression. For example,

$$\text{size}(P) = 2 \vee 3 \vee 4$$

can be written

$$\text{size}(P) = 2\,..\,4\,.$$

The negation of a selector,

$$\sim(\text{Term}_1 = \text{Term}_2)\,, \tag{A.19}$$

can be equivalently written

$$\text{Term}_1 \neq \text{Term}_2\,. \tag{A.20}$$

An arbitrary predicate $P(t_1, t_2, \ldots)$ can be written in the form of a referential selector

$$P(t_1, t_2, \ldots) = \text{True}\,.$$

Therefore, for the uniformity of terminology, a predicate will be considered a special form of a selector.

To facilitate the interpretation and readability of individual selectors in expressions, they are usually surrounded with square brackets and their conjunction is expressed by concatenating the bracketed forms (see Section 7).

APC expressions are created from selectors (relational statements) in the same way as predicte calculus expressions are created from predicates, i.e., by using logic connectives ($\sim$, &, $\vee$, $\Rightarrow$, $\Leftrightarrow$) and quantifiers. One additional useful connective is the *exception operation* ($\leftthreetimes$), defined as

$$S_1 \leftthreetimes S_2 \models (\sim S_2 \Rightarrow \sim S_1)\,, \tag{A.21}$$

where S_1 and S_2 are APC expressions. (S_1 S_2 reads: S_1 *except when* S_2). It is easy to see that the exception operator is equivalent to the symmetrical difference. In addition to ordinary quantifiers there is also a *numerical quantifier*, expressed in the form

$$\exists(I)v, S[v]\,, \tag{A.22}$$

where I, the *index set*, denotes a set of integers, and $S[v]$ is an APC expression having v as a free variable.

Sentence (A.22) evaluates as true if the number of values of v for which expression $S[v]$ is true is an element of the set I. For example, formula

$$\exists(2\,..\,8)v, S[v] \tag{A.23}$$

states that there are two to eight values of v for which the expression $S[v]$ is true. The following equivalences hold

$$\exists v, S[v] \quad \text{is equivalent to} \quad \exists(\geqslant 1)v, S[v]$$

and

$$\forall v, S[v] \quad \text{is equivalent to} \quad \exists(k)v, S[v],$$

where k is the number of possible values of variable v.

To state that there are k and only k distinct values for variables $v_1, v_2, \ldots, v_k$ for which expression $S(v_1, v_2, \ldots, v_k)$ is true we write:

$$\exists.v_1, v_2, \ldots, v_k, \; S(v_1, \ldots, v_k). \tag{A.24}$$

For example, the expression

$$\exists.P_0, P_1, P_2, [\text{contains}(P_0, P_1, \& P_2)] \,\&\, [\text{color}(P_1 \,\&\, P_2) = \text{red}] \Rightarrow [\text{two\_red\_parts}(P_0)]$$

states that predicate two_red_parts(P_0) holds if P_0 has two and only two distinct parts in it that are red.

Section 7 presents an example of the usage of the APC for formulating ovservational statements and concept descriptions.

ACKNOWLEDGMENT

In the development of the ideas presented here the author benefited from discussions with Tom Dietterich and Robert Stepp. Proofreading and comments of Jaime Carbonell, Bill Hoff, and Tom Mitchell were helpful in shaping up the final version of the paper.

The author gratefully acknowledges the partial support of the research by the National Science Foundation under grants MCS 79-06614 and MCS 82-05166.

REFERENCES

1. Amarel, F., An approach to automatic theory formation, in: von Foerster, H., Ed., *Illinois Symposium on Principles of Self-Organization* (1960).
2. Banerji, R.B., The description list of concepts, *Comm. ACM* **5** (1962) 426–431.
3. Banerji, R.B., *Artificial Intelligence: A Theoretical Perspective* (North-Holland, Amsterdam, 1980).
4. Bierman, A.W. and Feldman, J., Survey of results in grammatical inference, in: *Frontiers of Pattern Recognition* (Academic Press, New York, 1972) 32–54.
5. Bierman, A.W., The inference of regular LISP programs from examples, *IEEE Trans. Systems Man Cybernet.* **8**(8) (1978) 585–600.
6. Bongard, M.M., *Pattern Recognition* (Spartan Books, Washington, DC, 1970) [in Russian].
7. Brachman, R.T., On the epistomological status of semantic networks, Rept. No. 3807, AI Department, Bolt, Beranek and Newman, 1978.
8. Bruner, J.S., Goodnow, J. and Austin, G., *A Study of Thinking* (Wiley, New York, 1956).
9. Buchanan, G.B. and Feigenbaum, E.A., "Dendral and Meta-Dendral, their applications dimension, *Artificial Intelligence* **11** (1978) 5–24.

10. Buchanan, B.G., Mitchell, T.M., Smith, R.G. and Johnson, C.R., Jr., Models of learning systems, Tech. Rept. STAN-CS-79-692, Computer Science Department, Stanford University, 1979.
11. Burstall, R.M. and Darlington, J., A transformation system for developing recursive programs, *J. ACM* **24**(1) (1977) 44–67.
12. Carnap, R., The aim of inductive logic, in: Nagel, E., Suppes, P. and Tarski, A., Eds., *Logic, Methodology and Philosophy of Science* (Stanford University Press, Stanford, 1962) 303–318.
13. Case J. and Smith, C., Comparison of identification criteria for mechanized inductive inference, Tech. Rept. No. 154, State University of New York at Buffalo, 1979.
14. Chang, C. and Lee, R.C., *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, New York, 1973).
15. Cohen, B.L., A powerful and efficient structural pattern recognition system, *Artificial Intelligence* **9**(3) (1977) 233–255.
16. Coulon, D. and Kayser, D., Learning criterion and inductive behavior, *Pattern Recognition* **10**(1) (1978) 19–25.
17. Davis, R. and Lenat, D., *Knowledge-based Systems in Artificial Intelligence* (McGraw-Hill, New York, 1982).
18. Dietterich, T., Description of inductive program INDUCE 1.1, Internal Rept., Department of Computer Science, University of Illinois at Urbana-Champaign, 1978.
19. Dietterich, T., A methodology of knowledge layers for inducing descriptions of sequentially ordered events, Rept. No. 80–1024, Department of Computer Science, University of Illinois at Urbana-Champaign, 1980.
20. Dietterich, T. and Michalski, R.S., Inductive learning of structural descriptions: evaluation criteria and comparative review of selected methods, *Artificial Intelligence* **16**(3) (1981) 257–294.
21. Feigenbaum, E.A., The simulation of verbal learning behavior, in: Feigenbaum, E.A. and Feldman, J., Eds., *Computers and Thought* (McGraw-Hill, New York, 1963).
22. Fikes, R.E., Hart, R.E. and Nilsson, N.J., Learning and executing generalization robot plans, *Artificial Intelligence* **3** (1972) 251–288.
23. Gaines, B.R., Maryanski's grammatical inferences, *IEEE Trans. Comput.* **28** (1979) 62–64.
24. Gaschnig, J., Development of uranium exploration models for prospector consultant system, Artificial Intelligence Center, SRI Intern., 1980.
25. Hass, N. and Hendrix, G.G., An approach to applying and acquiring knowledge, *Proc. First Amer. Assoc. for AI Conference* (1980) 235–239.
26. Hájek, P., Havel, I. and Chytil, M., The GUHA method of automatic hypothesis determination, *Computing* **1**, (1966) 293–308.
27. Hájek, P. and Havránek, T., *Mechanizing Hypothesis Formation, Mathematical Foundations for a General Theory* (Springer, Berlin, 1978).
28. Hayes-Roth, F., A structural approach to pattern learning and the acquisition of classificatory power, *Proc. First Internat. Joint Conference on Pattern Recognition*, Washington, DC, October 30–November 1 (1973) 343–355.
29. Hayes-Roth, F. and McDermott, J., An interference matching technique for inducing abstractions, *Comm. ACM* **21**(5) (1978) 401–411.
30. Hedrick, C.L., A computer program to learn production systems using a semantic net, Ph.D. Thesis; Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1974.
31. Hintzman, D.L., *The Psychology of Learning and Memory* (Freeman, San Francisco, CA, 1978).
32. Hoff, B., Michalski, R.S. and Stepp, R., INDUCE 2—a program for learning structural descriptions from examples, Intelligent Systems Group Rept. No. 83-1, Department of Computer Science, University of Illinois at Urbana-Champaign, 1983.
33. Hovland, C.I., A 'communication analysis' of concept learning, *Psychol. Rev.* (1952) 461–472.
34. Hunt, E.B., *Concept Learning: An Information Processing Problem* (Wiley, New York, 1962).

35. Hunt, E.B., Marin, J. and Stone, P.T., *Experiments in Induction* (Academic Press, New York, 1966).
36. Jouannaud, J.P. and Kodratoff, Y., An automatic construction of LISP programs by transformations of functions synthesized from their input–output behavior, *Internat. J. Policy Anal. Inform. Systems* **4**(4) (1980) 331–358.
37. Kemeni, T.G., The use of simplicity in induction, *Psychol. Rev.* **62**(3) (1953) 391–408.
38. Kochen, M., Experimental study of hypothesis-formation by computer, in: Cherry, C., Ed., *Information Theory*, 4th London Symposium (Butterworth, London, 1961).
39. Langley, P., Neches, R., Neves, D. and Anzai, Y., A domain-independent framework for learning procedures, *Internat. J. Policy Anal. Inform. Systems* **4**(2) (1980) 163–198.
40. Larson, J., Inductive inference in the variable-valued predicate logic system VL_{21}: methodology and computer implementation, Ph.D. Thesis, Rept. No. 869, Department of Computer Science, University of Illinois, Urbana, Illinois, 1977.
41. Larson, J. and Michalski, R.S., Inductive inference of VL decision rules, *Proc. Workshop on Pattern-Directed Inference Systems*, Honolulu, Hawaii, May 23–27, 1977, *SIGART Newsletter* **63** (1977).
42. Lenat, D., AM: an artificial intelligence approach to discovery in mathematics as heuristic search, Computer Science Department, Rept. STAN-CS-76-570, Stanford University, Stanford, CA, 1976.
43. Lenat, D. and Harris, G., Designing a rule system that searches for scientific discovery, in: Waterman D.A. and Hayes-Roth, F., Eds., *Pattern-Directed Inference Systems* (Academic Press, New York, 1978) 25–51.
44. Michie, D., New face of artificial intelligence, *Informatics* **3** (1977) 5–11.
45. Michalski, R.S., A variable-valued logic system as applied to picture description and recognition, in: Nake F. and Rosenfeld, A., Eds., *Graphic Languages* (North-Holland, Amsterdam 1972) 20–47.
46. Michalski, R.S., AQVAL/1—computer implementation of a variable-valued logic system and its application to pattern recognition, *Proc. First Internat. Joint Conf. on Pattern Recognition*, Washington, DC, October 30–November 1 (1973).
47. Michalski, R.S., Variable-valued logic and its applications to pattern recognition and machine learning, in: Rine, D., Ed., *Multiple-Valued Logic and Computer Science* (North-Holland, Amsterdam, 1975).
48. Michalski, R.S., Synthesis of optimal and quasi-optimal variable-valued logic formulas, *Proc. 1975 Intern. Symposium on Multiple-Valued Logic*, Bloomington, IN, May 13–16, (1975) 76–87.
49. Michalski, R.S., Pattern recognition as rule-guided inductive inference, *IEEE Trans. Pattern Anal. Machine Intelligence* (1980).
50. Michalski, R.S., Knowledge acquisition through conceptual clustering: a theoretical framework and an algorithm for partitioning data into conjunctive concepts, *Internat. J. Policy Anal. Inform. Systems* **4**(3) (1980) 219–244.
51. Michalski, R.S. and Chilausky, R.L., Learning by being told and learning from examples, *Internat. J. Policy Anal. Inform. Systems* **4**(2) (1980) 125–160.
52. Michalski, R.S. and Larson, J.B., Selection of most representative training examples and incremental generation of VL_1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11, Rept. No. 78–867, Department of Computer Science, University of Illinois at Urbana-Champaign, 1978.
53. Michalski, R.S. and P. Negri, An experiment on inductive learning in chess end games, in: Elcock, E.W. and Michie D., Eds., *Machine Representation of Knowledge, Machine intelligence* **8** (Ellis Horwood, 1977) 175–192.
54. Michalski, R.S., Stepp, R. and Diday, E., A recent advance in data analysis: clustering objects into classes characterized by conjunctive concepts, in: Kanal, L. and Rosenfeld, A., Eds., *Progress in Pattern Recognition*, Vol. 1 (North-Holland, Amsterdam, 1981).
55. Minsky, M., A framework for representing knowledge, MIT AI Memo 306, 1974.

56. Mitchell, T.M., Version spaces: an approach to concept learning, Ph.D. Thesis, Stanford University, Stanford, CA 1978.
57. Mitchell, T.M., Generalization as search, *Artificial Intelligence* **18** (2) (1982) 203–226.
58. Moraga, C., A didactic experiment in pattern recognition, Rept. AIUD-PR-8101, Department of Informatics, Dartmund University, 1981.
59. Morgan, C.G., Automated hypothesis generation using extended inductive resolution, *Advance Papers 4th Internat. Joint Conf. on Artificial Intelligence*, Tbilisi, G.A., Vol. I (1975) 352–356.
60. Newell, A., Shaw, J.C. and Simon, H.A., A variety of intelligent learning in the general problem solver, Rand Corp. Tech. Rept. (1959) 1791.
61. Niblett, T. and Shapiro, A., Automatic induction of classification rules for a chess endgame, MIP-R-129, Machine Intelligence Research Unit, University of Edinburgh, 1981.
62. Nilsson, N.T., *Principles of Artificial Intelligence* (Tioga, Palo Alto, CA, 1980).
63. O'Rorke, P., A comparative study of inductive learning systems AQ11 and ID3, Intelligent Systems Group Rept. No. 81–14, Department of Computer Science, University of Illinois at Urbana-Champaign, 1981.
64. Pettorossi, A., An algorithm for reducing memory requirements in recursive programs using annotations, Internat. Workshop on Program Construction, Bonas, September 8–12, 1980.
65. Plotkin, G.D., A further note on inductive generalization, in: Beltzer, B. and Michie, D., Eds., *Machine Intelligence* **6** (Elsevier, New York, 1971).
66. Pokorny, D., Knowledge acquisition by the GUHA method, *Internat. J. Policy Anal. Inform. Systems* **4**(4) (1980) 379–399.
67. Polya, G., Mathematics and plausible reasoning, vol. I: induction and analogy in mathematics, Vol. II: patterns of plausible inference, (Princeton University Press, Princeton, NJ, 1954).
68. Popper, K.R., *The Logic of Scientific Discovery* (Basic Books, New York, 1959).
69. Post, H.R., Simplicity of scientific theories, *British J. Philos. Sci.* **11**(41) (1960).
70. Quinlan, J.R., Discovering rules by induction from large collections of examples, in: Michie, D., Ed., *Expert Systems in the Microelectronic Age* (Edinburgh University Press, Edinburgh, 1979).
71. Russell, B., *History of Western Philosophy* (Allen and Unwin, London, 1946) 566.
72. Sammut, C., Learning concepts by performing experiments, Ph.D. Thesis, Department of Computer Science, University of South Wales, Australia, 1981.
73. Shapiro, E.Y., Inductive inferences of theories from facts, Research Rept. 192, Department of Computer Science, Yale University, New Haven, CT, 1981.
74. Shaw, D.E., Swartout, W.R. and Green, C.C., Inferring LISP programs from examples, *Proc. 4th Internat. Joint Conf. on Artificial Intelligence*, Tbilisi, GA, Vol. I (1975) 351–356.
75. Shortliffe, E.H., *Computer-based Medical Consultations: MYCIN* (American Elsevier, New York, 1979).
76. Simon, H.A. and Kotovsky, Human acquisition for sequential patterns, *Psychol. Rev.* **10**(6) (1963) 534–546.
77. Simon, H. A. and Lea, G., Problem solving and rule induction: a unified view, in: Gregg, L.W., Ed., *Knowledge and Cognition*, Erlbaum, Potomac, MD, 1974).
78. Simon, H.A., *Models of Discovery* (Reidel, Dordrecht, 1977).
79. Smith, D.R., A survey of the synthesis of LISP programs from examples, Internat. Workshop on Program Construction, Bonas, September 8–12, 1980.
80. Solomonoff, R.J., A formal theory of inductive inference, *Inform. and Control* **7**, (1964) 1–22, 224–254.
81. Soloway, E.M. and Riseman, E.M., Levels of pattern description in learning, *Papers 5th Internat. Joint Conf. on Artificial Intelligence*, Cambridge, MA (1977) 801–811.
82. Stepp, R., The investigation of the UNICLASS inductive program AQ7UNI and user's guide, Rept. No. 949, Department of Computer Science, University of Illinois at Urbana-Champaign, 1978.
83. Stoffel, J.C., The theory of prime events: data analysis for sample vectors with inherently discrete variables, *Information Processing 74* (North-Holland, Amsterdam, 1974) 702–706.

84. Suppes, P., *Introduction to Logic* (Van Nostrand, Princeton, NJ, 1957).
85. Vere, S.A., Induction of concepts in the predicate calculus, *Advance Papers 4th Internat. Joint Conf. on Artificial Intelligence*, Tbilisi, GA, Vol. I (1975) 351–356.
86. Waterman, DA., Generalization learning techniques for automating the learning of heuristics, *Artificial Intelligence* **1**(1/2) (1970) 121–170.
87. Winston, P.H., Learning structural descriptions from examples, Tech. Rept. AI TR-231, MIT AI Lab, Cambridge, MA, 1970.
88. Winston, P.H., *Artificial Intelligence* (Addison-Wesley, Reading, MA, 1977).
89. Yau K.C. and Fu, K.S., Syntactic shape recognition using attributed grammars, *Proc. 8th Annual EIA Symposium on Automatic Imagery Pattern Recognition*, 1978.
90. Zagoruiko, N.G., Methods for revealing regularities in data, *Izd. Nauka* (1981) [in Russian].

Received August 1980; revised version received July 1982