INDUCTIVE INFERENCE BY REFINEMENT

P. D. Laird*
Department of Computer Science
Yale University
New Haven, Ct., 06520

Abstract

A model is presented for the class of inductive inference problems that are solved by *refinement* algorithms – that is, algorithms that modify a hypothesis by making it more general or more specific in response to examples. The separate effects of the syntax (rule space) and semantics, and the relevant orderings on these, are precisely specified. Relations called *refinement operators* are defined, one for generalization and one for specialization. General and particular properties of these relations are considered, and algorithm schemas for top-down and bottom-up inference are given. Finally, difficulties common to refinement algorithms are reviewed.

Introduction

The topic of this paper is the familiar problem of inductive learning: determining a rule from examples. Humans exhibit a striking ability to solve this problem in a variety of situations – to the extent that it is difficult to believe that a separate algorithm is at work in each case. Hence, in addition to the problems of implementing real systems that learn by example, there is the challenge of identifying fundamental principles that underlie this sort of learning.

As an illustration of the basic ideas, consider the following concept-learning problem (adapted from [Mitchell, 1982]). Objects have three attributes: size (large, small), color (red, yellow, blue), and shape (triangle, circle). A concept consists of an ordered pair of objects, possibly with some attributes left undetermined. For example, C = {(large ? circle), (small ? ?)} represents the concept "a large circle of any color, and any small object". There is a most-general concept ({(? ? ?),(? ? ?)}) and a large number (144) of most-specific concepts (both objects fully specified). Examples, or training instances, can be *positive* or *negative*: {(large blue circle), (small blue triange)} is a positive example of the concept C above, whereas {(large red triangle), (large blue circle)} is a negative example. If the current hypothesis excludes a positive example, the inference procedure must generalize it; and if it includes a negative example, the procedure must make it more specific. Every domain has rules for making a hypothesis more or less general; here, a concept can be generalized by changing an attribute from a specific value to '?', or specialized by the inverse operation.

*Work funded in part by the National Science Foundation, under Grants MCS8002447 and DCR8404226.

The essential features of this simple illustration apply to many inductive learning problems in a variety of domains: formal languages and automata (e.g., [Angluin, 1982], [Crespi-Reghizzi, 1972]); programming languages (e.g., [Hardy, 1975], [Shapiro, 1982], [Summers, 1977]); functions and sequences ([Hunt et al., 1966], [Langley, 1980]); propositional and predicate logic ([Michalski, 1975], [Shapiro, 1981], [Valiant, 1984]), and a variety of representations specific to a particular domain (e.g., [Feigenbaum, 1963], [Winston, 1975]).

From the experiences of many researchers (see, for example, [Angluin and Smith, 1983], [Banerji, 1985], [Cohen, 1982], [Michalski, 1983] for summaries), a number of general guidelines have been suggested:

- Define a space of examples and a space of rules rich enough to explain any set of examples.
- Given some examples and a set of possible hypotheses, generalize the hypotheses that fail to explain positive examples, and specialize hypotheses that imply negative examples.
- If possible, represent examples in the same language used to express the rules.

Our goal is to present a formal model to unify many of the ideas common to these domains. The value of such a formalism is that the essential features of the inductive component of a projected application can be identified quickly, and basic algorithms constructed, without the need to rediscover these ideas from first principles. Another advantage is that the abstract properties and limitations common to algorithms based on this model can be identified and studied without reference to the details of a particular application.

This report is necessarily brief, with only the outlines of the principal concepts given, plus examples to illustrate their application. More details, examples, and proofs are available in the full report ([Laird, 1985]).

Inductive Inference Problems

Definition 1 An *inductive inference problem* has six components:

- A partially ordered set $(\mathcal{D}, \geq)$, called the *semantic domain.*
- A set $\mathcal{E}$ of expressions over a finitely presented algebra, called the *syntactic domain.*

- A mapping h: $\mathcal{E} \rightarrow \mathcal{D}$ such that every d in $\mathcal{D}$ is $h(e)$ for some expression e in $\mathcal{E}$.
- A designated element d_0 of $\mathcal{D}$, called the *target object.*
- An oracle, EX, for "examples" of d_0, in the form of signed expressions in $\mathcal{E}$. If $EX()$ returns $+e$, then $d_0 \geq h(e)$, and if $EX()$ returns $-e$, then $d_0 \not\geq h(e)$.
- An oracle ($\geq$?) for the partial order, such that ($e_1 \geq e_2$?) returns 1 if $h(e_1) \geq h(e_2)$, and 0 otherwise.

The examples below will help this definition seem less abstract. It is more general than most definitions of inductive inference, in that target objects need not be subsets of some set; instead, they are simply elements of a partial ordering. Rules are expressions over some algebra which is expressive enough to name every possible target object. The mapping h is the association between rules and the objects they denote. Note that h may map more than one expression to the same semantic element; if $h(e_1) = h(e_2)$, then e_1 and e_2 are called *h-equivalent* rules. There may, of course, be different syntactical representations of one semantic domain (grammars, automata, logical axioms, etc.); according to this model, the problem changes when a new syntax $\mathcal{E}$ is adopted.

Examples are elements of the same set $\mathcal{E}$ of expressions: i.e., every expression in $\mathcal{E}$ is potentially an example – positive in case its semantic representation is no greater than than the target, and negative otherwise. In practice, the set of examples is often limited to a subset of the expressions (see below). The oracle EX represents the mechanism which produces examples of the target; in actuality, examples may come from a teacher, a random source, a query-answering source, or combinations of these. Note that EX depends on the target object d_0.

The oracle ($\geq$?) serves to abstract the aspect of the problem concerned with testing whether an expression imples an example. In practice, the complexity of this problem ranges from easy to unsolvable. By oracularizing it we are choosing to ignore the complexity of this problem while we study the abstract properties of inductive inference.

Example 1 Let $X = \{x_1, \ldots, x_t\}$ be a finite set, and $\mathcal{D} = 2^X$, the set of subsets of X. For example, X might denote a set of automobile attributes (sedan, convertible, front-wheel drive, etc.), while an element of $\mathcal{D}$ specifies those attributes possessed by a particular model. Let $\mathcal{D}$ be partially ordered by $\supseteq$, the containment relation. There are many possible languages for representing $\mathcal{D}$, such as the conventional one for set theory. A more algebraic language is a Boolean algebra over the elements $x_1, \ldots, x_t$ with elements of $\mathcal{D}$ represented by monomials of degree t (*minterms*). Thus the empty set is represented by $x'_1 \ldots x'_t$ (x' denotes the complement of x), and $\{x_1\} = h(x_1 x'_2 x'_3 \ldots x'_t)$. In this case h is a bijection between minterms and elements of $\mathcal{D}$. If m_1 and m_2 are minterms, then m_1 is a positive example of m_2 iff every uncomplemented variable in m_1 occurs uncomplemented in m_2.

Example 2 Let $\mathcal{D}$ be the class of partial recursive functions mapping integers to integers. If f_1 and f_2 are functions in $\mathcal{D}$, define $f_1 \geq f_2$ iff for every integer x such that $f_2(x)$ is defined, $f_1(x) = f_2(x)$. A convenient language for representing the functions in $\mathcal{D}$ is the subset of LISP expressions mapping integers to integers. If P is such a program, then $h(P)$ is the partial function computed by P. Consider the function $f(x) = x^2$. A positive example of f is the program `(LAMBDA (X) (COND ((= X 2) 4)))`. Intuitively, this example states that $f(2) = 4$ without giving any other values of f. The problem of deciding for two arbitrary programs P_1 and P_2 whether $P_1 \geq P_2$ is, of course, recursively unsolvable.

The general inductive inference problem allows any expression in $\mathcal{E}$ to be an example. More often we are limited to a subset of $\mathcal{E}$ for examples (e.g., in Example 2 above, we may be given only programs defined on a single integer rather than arbitrary programs). But what property guarantees that a subset of $\mathcal{E}$ is sufficient to identify a target uniquely?

Definition 2 Let S be a set of examples of d_0. An expression e is said to *agree with* S if for every positive example e^+ in S, $h(e) \geq h(e^+)$, and for every negative example e^- in S, $h(e) \not\geq h(e^-)$.

Definition 3 A *sufficient set of examples* of d_0 is a signed subset S of $\mathcal{E}$ with the property that all expressions that agree with S are h-equivalent and are mapped by h to d_0.

Example 3 In Example 1 above, a sufficient set of examples for any target can be formed from only the t minterms with exactly one uncomplemented variable. In Example 2, a sufficient set of examples can be constructed from programs of the form: `(LAMBDA (X) (COND ((= X i) j)))`, where i and j are integer constants.

Example 4 In many concept-learning problems, objects possess a subset of some group of t binary attributes, and a concept is a subset of the set of all possible distinct objects. A "ball", for example, might denote the concept consisting of all objects with the "spherical?" attribute, regardless of other attributes ("red?", "wooden?", etc.). As a formal expression of this domain, let $\{x_1, \ldots, x_t\}$ be Boolean variables, and $\mathcal{D}$ the set of sets of assignments of values (0 or 1) to all of the variables, partially ordered by $\supseteq$. For syntax we may take the set of Boolean expressions; h maps an expression to the set of satisfying assignments. Expression e_1 is an example of e_2 iff the Boolean expression "$e_1 \rightarrow e_2$" is a tautology ($\rightarrow$ denotes implication), since then any assignment satisfying e_1 then must also satisfy e_2. A sufficient set of examples for any expression can be formed from the set of minterms, since these represent a single assignment.

Finally, note that for every inductive inference problem there is a dual problem, differing only in that $\mathcal{D}$ is partially ordered by $\leq$ (rather than $\geq$). Then e_1 is a positive example of e_2 if $e_2 \leq e_1$, and a negative example otherwise.

Refinements

In most applications, the mapping h: $\mathcal{E} \rightarrow \mathcal{D}$ is not just an unstructured assignment of expressions to objects. Usually there

is an ordering $\succeq$ of the expressions that is closely related to that on the underlying semantics. For example, referring again to the size-color-shape example in the introduction, we see that the syntactic operation of replacing an attribute ("red") by the don't-care token ("?") corresponds semantically to replacing the set of expressed objects by a larger set. The ordering $\succeq$ may only be a quasi-ordering (reflexive and transitive but not anti-symmetric). But we can still use it to advantage in computing generalizations and specializations of hypotheses, provided it has three properties: an order-homomorphism property with respect to h; a computational sub-relation called a *refinement*; and a completeness property. These we shall now define.

Definition 4 Let $\succeq$ be an ordering of $\mathcal{E}$. The mapping h: $\mathcal{E} \rightarrow \mathcal{D}$ is said to be an *order-homomorphism* if, for all e_1 and e_2 in $\mathcal{E}$ such that $e_1 \succeq e_2$, $h(e_1) \geq h(e_2)$.

Example 5 Referring back to Example 1, let $m_2 \succeq m_1$ if every uncomplemented variable in m_1 also occurs uncomplemented in m_2. Then h is an order-homomorphism.

In Example 2, it is not clear how to define $P_2 \succeq P_1$ on arbitrary LISP programs so as to form an order-homomorphism. In [Summers, 1977], this problem is solved by restricting the class of LISP programs to have a very specific form.

Example 6 Because of the expressiveness of predicate calculus, many systems use a first-order language $\mathcal{L}$ (or subset thereof) as the syntactic component of the inference problem. But what is the semantic component, and how is it ordered? Example 4 is the analogous case for propositional logic, where $\mathcal{D}$ consisted of *sets* of assignments, ordered by $\supseteq$. With first-order logic, Herbrand models (i.e., sets of variable-free atomic formulas) take the place of assignments, and $\mathcal{D}$ consists of classes of (first-order definable) Herbrand models, ordered by $\supseteq$. The sentence $\forall x\,(red(x) \vee \sim large(x))$ designates the class of models in which everything that is large is also red.

A syntactic ordering is as follows: given sentences φ_1 and φ_2 in $\mathcal{L}$, define $\varphi_2 \succeq \varphi_1$ iff $\vdash \varphi_1 \rightarrow \varphi_2$ (where $\rightarrow$ indicates implication). It is easy to see that h is an order-homomorphism: if φ_1 implies φ_2, then any model of φ_1 is also a model of φ_2, and hence the models of φ_1 are a subset of those of φ_2.

The importance of $\succeq$ to inductive inference is as follows: Suppose a hypothesized rule e is found to be *too general* in the sense that there exists a negative example e^- such that $h(e) \geq h(e^-)$. Then (assuming h is an order-homomorphism) any new hypothesis e' such that $e' \succeq e$ will also be too general, and hence need not be considered. Similarly, if e is *too specific*, then any hypothesis e' such that $e \succeq e'$ can be eliminated from consideration.

In order to take advantage of the efficiency induced by the syntactic ordering $\succeq$, we need the means to take an expression and obtain from it expressions that are more general or more specific. This leads to the notion of a *refinement* relation.

Definition 5 An *upward refinement* γ for $\succeq$ is a recursively enumerable (r.e.) binary relation on $\mathcal{E}$ such that *(i)* γ^* (the reflexive-transitive closure of γ) is the relation $\succeq$; and *(ii)* for all $e_1, e_2 \in \mathcal{E}$, if $(e_1, e_2) \in \gamma$ then $h(e_1) \geq h(e_2)$. The notation $\gamma(e)$ denotes the set of expressions e_1 such that $(e_1, e) \in \gamma$.

There is a dual definition of a *downward refinement* ρ for an ordering $\preceq$: $\rho^* = \preceq$, and if $(e_1, e_2) \in \rho$ then $h(e_1) \leq h(e_2)$. Nearly everything true of upward refinements has a dual for downward refinements, but to save space we shall omit dual statements.

The r.e. condition on refinements means that they can be effectively computed. Thus if e is found to be too specific, an inference algorithm can compute the set $\gamma(e)$, and γ of each of these expressions, etc., in order to find a more general hypothesis. We would like to know that, by continuing to refine e upward in this way, we will eventually obtain an expression for every object d more general than $h(e)$. This motivates the *completeness* property of refinements.

Definition 6 An upward refinement γ is said to be *complete for* $e \in \mathcal{E}$ if $h(\gamma^*(e)) = \{d \mid d \geq h(e)\}$. If γ is complete for all $e \in \mathcal{E}$ then γ is said to be *complete*.

Example 7 In the language of Example 1, let $\gamma(m)$ be the set of minterms m' obtained from m by uncomplementing exactly one of the complemented attributes. Thus $\gamma(x_1' x_2' \ldots x_t') = \{(x_1 x_2' \ldots x_t'), (x_1' x_2 x_3' \ldots x_t'), \ldots, (x_1' \ldots x_{t-1}' x_t)\}$, and $\gamma(x_1 x_2 \ldots x_t) = \emptyset$. It is easily seen that γ is a complete upward refinement for minterms. A complete downward refinement for minterms $\gamma(m)$ computes the set of minterms obtained from m by complementing one of the uncomplemented attributes.

Example 8 Let $\mathcal{E}$ be the set of first-order clauses (i.e., disjunctions of atomic literals or their negation) with only universal quantification, assuming some fixed language $\mathcal{L}$. An upward refinement for $\mathcal{E}$ (with respect to the ordering $\succeq$ of Example 6) is as follows ([Shapiro, 1981]):

Let C be a clause. $\gamma(C)$ is the set of clauses obtained from C by applying exactly one of the following operations:

- Unify two distinct variables x and y in C (replace all occurrences of one by the other).
- Substitute for every occurrence of a variable x a most-general term (i.e., function call or constant) with fresh variables. (For example, replace every x by $f(x_1)$ where f is a function symbol in $\mathcal{L}$ and x_1 does not occur in C.)
- Disjoin a most-general literal with fresh variables (i.e., $p(x_1)$ or $\sim p(x_1)$, where p is a predicate symbol and x_1 does not occur in C).

For example, let $r(x, y)$ stand for the relation x *is-a-blood-relative-of* y, $f(x)$ for the function *the-father-of* x, and $m(x)$ for the function *the-mother-of* x. Let C be the clause $r(x, f(y)) \rightarrow r(x, y)$, meaning that if someone is related to a person's father, then he is related to that person. The following clauses are all in $\gamma(C)$:

- $r(x, f(x)) \rightarrow r(x, x)$
- $r(m(z), f(y)) \rightarrow r(m(z), y)$
- $r(x, f(y)) \rightarrow (r(x, y) \vee r(z_1, z_2))$

Each of the derived clauses is easier to satisfy and hence has more models.

More examples of refinements over various domains are given in [Laird, 1985]. The task of constructing a refinement can be tricky, because one must ensure that all useful generalizations or specializations have been included. But given the formal definition, it is basically a problem in algebra, rather than a heuristic problem as has usually been the case for most applications. In essence, the refinement is a formal expression of the "production rules" or "generalization rules" found in many implementations (e.g., [Michalski, 1980], [Mitchell, 1977]).

Below we sketch a simple "bottom-up" algorithm for inductive inference using an upward refinement. For simplicity we assume that

1. $\gamma(e)$ is finite for all e. (γ is then said to be *locally finite.*)
2. There is an expression $e_{min} \in \mathcal{E}$ such that $e \succeq e_{min}$ for all $e \in \mathcal{E}$.
3. γ is complete for e_{min}.

The algorithm repeatedly calls on EX and tests the current hypothesis against the resulting example. If the hypothesis is too specific, it refines it, placing the more general expressions onto a queue and taking a new hypothesis from the front of the queue. If the hypothesis is too general, it discards it (with no refinement) and takes a new one from the queue. It can be shown that the algorithm will eventually converge to a correct hypothesis provided EX presents a sufficient set of examples for the target.

```
ALGORITHM UP_INFER:
Initialize H ← e_min.
           QUEUE ← empty().
           EXAMPLES ← empty().
Do Forever:
  EXAMPLES ← EXAMPLES ∪ {EX()}.
  While H disagrees with any EXAMPLES:
    Using the (≥?) oracle, check
       that H ≥ no negative examples, and
       H ≱ some positive example.  If so,
       add γ(H) to QUEUE.
    H ← front(QUEUE).
```

By duality we can construct a top-down algorithm using γ and e_{max}. Other algorithms are also possible, depending on the properties of the domain and the refinements. Most inductive inference algorithms in the literature are either top-down or bottom-up ([Mitchell, 1982] suggests using both in parallel). And for some domains, one direction seems advantageous over the other (conjunctive logical domains, for example, seem to prefer generalization to specialization). This directional asymmetry seems to occur mainly when the refinement is locally finite in one direction but not in the other. Note that this is a *syntactic* property, not a semantic one: regular sets of strings, for example, are easier to infer bottom-up when the rules are expressed as automata or as logical axioms ([Biermann and Feldman, 1972], [Shapiro, 1981]), but top-down when the rules are expressed as regular expressions ([Laird, 1985]).

Finally, it is worth observing how refinement algorithms handle the so-called Disjunction Problem. In the context of classical concept-learning, this refers to the problem of forming a "reasonable" generalization from examples in a domain that includes a disjunction operation. The trivial generalization, consisting of the disjunction of all the positive examples, is usually unsuitable since it will never converge to a hypothesis representing an infinite set. On the other hand, it is undesirable to eliminate the trivial generalization as a possibility, since it might lead to the correct rule.

Since refinement algorithms such as the one above apply the operator γ to hypotheses in the order in which they are discarded, the minimal generalization (adding only the one element to the set) is not necessarily the first one tried. For example, suppose the domain is the class of regular sets of strings over the alphabet $\{0, 1\}$ and examples are strings in and not in the target set. If the current hypothesis is represented by a regular expression R, and a string w_1 is presented that is not included by R, a refinement algorithm will generalize by applying γ to R, producing a set of new expressions to be tried in turn as hypotheses. Among these are expressions which extend R to $R + w_1$; but other expressions, such as R^*, will also be constructed and held on the queue in order. If another positive example w_2 is presented, $R + w_1$ will be discarded, but R^* will be considered before the refinements of $R + w_1$ (including the trivial one $R + w_1 + w_2$). It can be shown that the algorithm will converge to a correct expression, *whether or not the target is a finite set of strings.*

Limitations of the Refinement Approach

The induction-by-refinement model is not expected to yield efficient algorithms directly since it is too general to take advantage of specific properties of the domain. Instead, the primary value of the model is the way it clarifies the important roles played by the semantic and syntactic orderings, and in the definition of refinement operators for computing appropriate generalizations and specializations of hypotheses. Recently several researchers have been looking for efficient inference algorithms that yield (with high probability) rules whose "error" is arbitrarily small, as measured by the probability distribution governing the presentation of examples ([Valiant, 1984], [Valiant, 1985], [Blumer et. al., 1986]). In many of these algorithms, a refinement operation is clearly being employed; but instead of generating all refinements $\gamma(e)$, the examples are used to reduce the set of possibilities – e.g., $\gamma(e, x)$ is computed using the example x, yielding more general expressions that are consistent with x.

There are many domains in which the partial order is too linear or too "flat" to be of much use in searching for hypotheses. Consider, for example, the problem of finding an arithmetic recursion relation of the form $s_n = f(s_{n-1})$ to explain a sequence of integers. We might, for instance, try the hypothesis $s_n = s_{n-1}^2 + 5$ and find that it explains only one integer in the sequence. At this point, the "less defined than or equal to" ordering used in Example 2 is no more useful for finding a more general function than a simple generate-and-test approach.

Refinement algorithms have generally performed poorly when the examples are subject to "noise" (e.g., [Buchanan and Mitchell, 1978]). They also tend to require that all examples be stored, so that later refinements can be tested to avoid over-generalization or -specialization (e.g., [Shapiro, 1982]). These two limitations

are inevitably related: for, a procedure which is tolerant of faulty examples cannot expect to find a hypothesis consistent with every example seen so far and hence must be selective about the examples it chooses to retain. It is interesting to note that nearly all refinement algorithms in the literature refine *in only one direction* (up or down). Consequently these algorithms cannot recover if they over-refine in response to a faulty example. By contrast, an algorithm which can refine upward and downward has the potential for correcting an over-generalization resulting from a false positive example when subsequent examples so indicate (e.g., [Shapiro, 1982]).

Finally – and most serious – the refinement technique relies on a fixed algebraic language, without suggesting any way to incorporate new "terms" or "concepts" into the language. In the learning of geometric shapes, for example, we could in principle define complex patterns in terms of the elementary relations of the language (edge, circle, above, etc.), but the description would be too complex to find by searching through the rule space $\mathcal{E}$. By contrast, a suitable set of higher-level concepts (e.g., triangle, box, inside) could make the refinement path to a successful rule short enough to find by searching. But I am unaware of any general technique for discovering such new terms (other than having a friendly "teacher" present them explicity). A successful model of this process would be a significant advance in the study of inductive learning.

Acknowledgements

I am especially grateful to Dana Angluin for many helpful discussions of this work. Thanks also to Takeshi Shinohara, whose careful reading of the original report identified some errors and improved the exposition.

References

[1] Angluin, D. Inference of Reversible Languages. *J. ACM 29*:3 (1982) 741-765.

[2] Angluin, D. and C. H. Smith. Inductive Inference: theory and methods. *Computing Surveys 15*:3 (1983) 237-269.

[3] Banerji, Ranan. The logic of learning. In *Advances in Computers 24*, M. Yovits, ed., Orlando: Academic Press, 1985, pp. 177-216.

[4] Biermann, A. W. and J. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Comput. C-21*:6 (1972) 592 - 597.

[5] Blumer, A., A. Ehrenfeucht, D. Haussler, M. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonekis Dimension. *Proc. 18th ACM Symp. Theory of Comp.* May, 1986.

[6] Buchanan, B. G. and T. M. Mitchell. Model-Directed learning of production rules. In *Pattern-directed inference systems*, D. A. Waterman and F. Hayes-Roth, eds. New York: Academic Press, 1978, pp. 297-312.

[7] Cohen, P. R. and E. A. Feigenbaum, eds. *The handbook of artificial intelligence*, Vol. III. Los Altos: William Kaufmann, Inc., 1982.

[8] Crespi-Reghizzi, S. An effective model for grammar inference. In *Information Processing 71*, B. Gilchrist, ed. New York: Elsevier North-Holland, 1972, pp. 524-529.

[9] Feigenbaum, E. A. The simulation of verbal learning behavior. In *Computers and Thought*, E. A. Feigenbaum and J. Feldman, eds. New York: McGraw-Hill, 1963.

[10] Gold, E. M. Language identification in the limit. *Information and Control 10* (1967) 447-474.

[11] Hardy, S. Synthesis of LISP programs from examples. *Proc. IJCAI-75*, pp. 268-273.

[12] Hunt, E. B., J. Marin, and P. J. Stone. *Experiments in Induction*. New York: Academic Press, 1966.

[13] Laird, P. D. Inductive inference by refinement. Tech. Rpt. 376, Department of Computer Science, Yale University, New Haven, Ct., 1986.

[14] Langley, P. W. Descriptive discovery processes: experiments in Baconian science. Tech. Rep. CS-80-121, Computer Science Department, Carnegie-Mellon University, 1980.

[15] Michalski, R. Variable-valued logic and its applications to pattern recognition and machine learning. In *Computer Science and multiple-valued logic theory and applications*, D. Rine, ed. Amsterdam: North-Holland, 1975, pp. 506-534.

[16] Michalski, R. Pattern recognition as rule-guided inductive inference. *IEEE Trans. Pat. Anal. and Mach. Intel. (PAMI-2)*:4 (1980) 349-361.

[17] Michalski, R. Theory and methodology of inductive learning. *AI 20*:2 (1983) 111-161.

[18] Mitchell, T. M. Version Spaces: a candidate elimination approach to rule learning. *Proc. IJCAI-77*, pp. 305-310.

[19] Mitchell, T. M. Generalization as search. *Artificial Intelligence 18*:2 (1982) 203-226.

[20] Shapiro, E. Y. (1981). Inductive inference of theories from facts. Tech. Rep. 192, Department of Computer Science, Yale University, New Haven, Ct., 1981.

[21] Shapiro, E. Y. (1982). Algorithmic program debugging. Ph. D. dissertation, Computer Science Department, Yale University, New Haven, Ct., 1982. Published by M.I.T. Press.

[22] Summers, P. D. A methodology for LISP program construction from examples. *J.ACM 24*:1 (1977) 161-175.

[23] Valiant, L. G. A theory of the learnable. *C. ACM 27*:11 (1984) 1134-1142.

[24] Valiant, L. G. Learning disjunctions of conjunctions. *Proc. IJCAI-85*, pp. 560 - 566.

[25] Winston, P. H. Learning structural descriptions from examples. In *Psychology of Computer Vision*, P. H. Winston, ed. New York: McGraw-Hill, 1975.