I INTRODUCTION

Knowledge must be represented in abstract form in knowledge-based systems, so that it can be stored and manipulated effectively. Although experts have difficulty formulating their knowledge explicitly as rules and other abstractions, they find it easy to demonstrate their expertise in specific performance situations. Schemes for learning abstract representations—or concepts—from examples offer the potential for domain experts to interact directly with machines to transfer their knowledge. This potential is rarely realized in practice because the field of machine learning is in turmoil and few useful general principles have been articulated. This paper develops a framework for describing concept learning techniques which enables their relevance to particular knowledge engineering problems to be evaluated.

In recent years the study of machine learning has successfully elucidated some basic techniques for generalizing concrete examples to more abstract descriptions. These include heuristics for generalizing particular data types, candidate-elimination algorithms, methods for generating decision-trees and rule-sets, back propagation of constraints through an explanation tree, function induction, and synthesis of procedures from execution traces. Many interesting individual programs have been developed, but little rationalization, re-implementation, and replication of results. No coherent methodology has emerged for describing and categorizing learning techniques to make them readily accessible to potential users. In fact there are almost as many paradigms for machine learning as there are systems, and the variety of different, evocative, connotation-laden words used to describe simple mechanisms is one of the biggest problems in coming to grips with the field. Methods have been reported which claim to learn by *analogy, being told, debugging, discovery, doing, examples, experimentation, exploration, imitation, instruction, observation, rote,* and *taking advice.* The apparent richness and variety of these approaches may give a misleading impression of a field teeming with fruitful techniques, with a selection of well-defined methods for tackling any given problem. The present research is motivated by a desire to rationalize the plethora of approaches and paradigms, and provide a consistent organized structure on which system designers can base practical implementation decisions.

The framework developed in this paper aims to help users identify the most appropriate concept learning techniques for a given application. The next section discusses the meaning of concept learning and sets out its theoretical foundation in logic. From the standpoint of knowledge engineering, concept learning systems differ in their representation of concepts and examples, ways of biasing the search involved in determining a concept, and mode of teacher interaction; these are treated separately in the following three sections. Finally, several schemes for learning are reviewed and placed within the framework. Another perspective on the spectrum of machine learning techniques, concentrating more on classifying a wide variety of existing systems, can be found in [63].

At present, one can point to few success stories where concept learning has been used to build or enhance actual expert systems. Most research deals with "toy" problems, from Winston's ground-breaking program that learned the structure of an arch (supports, lintel, etc) from example arches [70] to more recent systems that learn empirical laws from experimental data [33,36]. An oft-cited success is Michalski's experiment on diagnosing soybean diseases, where the system reportedly found rules that were sometimes better (and in no case worse) than those supplied by human controls [47]. One practical example of the use of concept learning techniques for an expert system problem can be found in [15].

II CONCEPT LEARNING

Current knowledge acquisition (KA) systems perform routine housekeeping, permit rote learning of explicitly-presented facts, and are able to elicit from experts simple rules based on the attributes of example cases (eg [13]). But they cannot infer rules which operate on more powerful representations (eg nested structures), nor can they learn from cases presented by non-expert users. Methods of concept learning may be able to overcome these limitations, although the present state of the art is primitive and suggests ideas rather than well-developed algorithms for the KA toolbox. Concept learning systems take examples and create general descriptions, often expressed as rules, which expert systems need.

A *The meaning of "concept learning"*

Dictionary definitions of "concept" are remarkably vague, but have in common the abstract idea of a class of objects, particularly one derived from specific instances or occurrences. "Learning" is an equally broad term, and denotes the gaining of knowledge, skill, and understanding from instruction, experience, or reflection—in other words, "knowledge acquisition" by people. Notwithstanding the grand and sweeping nature of these ideas, the interpretation of concept learning within AI is much more constricted and specific. We take it to denote *the acquisition of structural descriptions from examples of what is being described.*

Others have defined terms such as "generalization" [62], "inductive learning" [46], and "inductive modeling" [11] in almost identical ways. Moreover, what is learned is sometimes called a "generalization", a "description", a "concept", a "model", a "hypothesis". A satisfactory technical vocabulary has not yet been developed; which term one favors seems purely a matter of taste.

Concept learning involves acquiring descriptions which make the structure of generalizations explicit. When a person learns a new concept, they gain knowledge that they can use in a rich variety of different ways: apply it to a particular case, reflect on it, tell it to a friend, criticize it, relate it to other ideas, and so on. For a person, learning is not simply a matter of acquiring a description, but involves taking something new and integrating it fully with existing thought processes. Computer programs which embody concept learning are not nearly so capable! However, they do acquire descriptions which are explicit in the sense that they can be communicated economically (for example, in the form of rules) and could plausibly support a variety of different kinds of reasoning. This orientation rules out, for example, connectionist models of learning, which embed knowledge in high-dimensional numerically-parametrized spaces, making learning a process of weight adjustment.

While the phrase may conjure up an alluring promise of the magic of human intelligence, systems for concept learning are rooted firmly in the reality of practical algorithms. The essence is a constrained search of what is invariably an astronomically large space of possible descriptions. The framework developed below is based on distinctions, summarized in Figure 1, which are classified into the three areas of representing concepts and examples (section III); biasing the search for concepts (section IV); and interacting with a teacher (section V).

B *Basis for concept learning*

Two kinds of concept learning are distinguished: *inductive* and *deductive*. For the first, one must infer a general conclusion from empirical examples. This can be formalized as follows [22]. Along with initial background knowledge BK, examples E of a desired concept are given. A concept C is said to be *induced* if:

BK	$\not\Longrightarrow$	E	The examples are not logical consequences of the knowledge by itself
$BK \cup E$	$\not\Longrightarrow$	$\neg C$	The concept is not inconsistent with the knowledge and examples
$BK \cup C$	$\Longrightarrow$	E	The examples are logical consequences of the concept and knowledge together

For example, in a playing-card domain the background knowledge BK might comprise facts about the value and suit of each card (numbered from 1 to 52). The concept `pair`, which is not included in the domain knowledge, might be induced from examples E of pairs of cards whose values are the same. It could be expressed as

C : `value(X)=value(Y)` $\Longrightarrow$ `pair(X,Y)`.

Since the `pair` predicate is not mentioned in BK, the first two conditions above are satisfied. The third is satisfied since (presumably) the examples E really are pairs.

Normally $BK \cup E \not\Longrightarrow C$ (ie C is induced rather than deduced). If E is known to be exhaustive, the learner may be able to deduce a concept. For example, in acquiring the concept `black-card`, if the examples include *every* playing card in the deck—with red ones as counterexamples—the concept can be deduced providing the learner knows that E is exhaustive. The inference of a statement from information which is known to exhaust all possibilities is a special case of induction known as *summative induction*, and lies on the borderline with deduction.

Suppose examples and concept are both deducible from BK, even though neither are explicitly present in it. This is an instance of *deductive* concept learning. Although excluded by the above formalization, it can be viewed as a kind of learning, for the examples show which deductions are important—namely those that represent the reasoning involved in the examples—and enable this important knowledge to be operationalized in an explicit description. In most cases an impossibly large set of operationalized concepts can be deduced from BK, and the problem is to select one appropriate to the examples. Deductive concept learning may be formalized as follows:

$BK \Longrightarrow E$	The examples are a logical consequence of the background knowledge
$C \notin BK$	The concept is not an explicit part of the background knowledge
$BK \Longrightarrow C$	The concept is a logical consequence of the background knowledge
$C \Longrightarrow E$	The examples are a logical consequence of the concept

In other words, the concept operationalizes the relevant part of BK. This kind of learning is often called "explanation-based" [49].

The distinction between deductive and inductive concept learning can be viewed as a modern reincarnation of the long philosophical tradition of distinguishing *necessary* from *contingent* truths. The former are facts which could not be otherwise, in other words they can be deduced from the domain theory; while the latter could be otherwise and hence cannot be deduced. This is often held to rest on the distinction, made by Kant, between *analytic* statements, which are tautologous, and *synthetic* ones that provide new information [26]. Many philosophers (notably Quine) have been sceptical of this distinction. If attention is paid to the *process* rather than just the *result* of reasoning, the seemingly sharp-edged analytic/synthetic boundary softens, and all statements can be viewed as synthetic to varying degrees. From a process viewpoint, noticing that 2670091735108484737 is a prime number, or that it is a factor of $3^{136} + 1$, is more than "just" exposing a tautology[1] [69].

The two kinds of concept learning, deductive and inductive, correspond to the blurring of analytic and synthetic statements. Both involve search, but whereas deductive methods express concepts in a language based on inferences from BK, inductive methods may use more general descriptions.[2] While inductive search usually seems even more open-ended than deductive search, in both cases the space is often infinite and in practice one's powers of both induction and deduction are limited by ability to search. The differences are twofold. Firstly, in induction, search is governed by a complexity ordering, while for deduction it terminates when a pre-defined goal is reached. Secondly, the object of interest—the concept learned—is the outcome of induction, while the result of deduction is (a generalization of) the search path itself.

To illustrate how regarding search as fundamental blurs the distinction between inductive and deductive learning, suppose it were possible to take the proof that 2670091735108484737 is prime, and generalize it in a way that allows us to decide easily whether other large integers are prime. That could be just as much a practically-useful discovery as inductively spotting some pattern common to the integers 2, 3, 5, 7, 11, $\cdots$.

Another issue which relates to the formal basis of concept learning concerns the ordering of concepts according to preference. (Section IV considers in more detail this and other ways in which the search can be biased.) The search may be arranged to consider candidate concepts in a predetermined order, terminating as soon as a suitable one is discovered. This is necessary whenever exhaustive search is infeasible. It subsumes notions such as model maximization [22], complexity ordering [21], and parsimony principles such as Occam's razor. Occam's razor defines an ordering—*always prefer the simplest concept*—but leaves open the question of how to measure complexity. Since simplicity/complexity is a property of predicates, such principles cannot be expressed formally in first-order logic; they require at least second-order logic.

III CONCEPT AND EXAMPLE REPRESENTATION

Concepts and the examples from which they are created are the output and input of the KA system; what is learned and what is provided by a teacher (or other external agent). To be useful, a framework for representing concepts must provide knowledge engineers with methods for selecting appropriate representations for examples, concepts and background knowledge. Separate (but related) representations are required for examples and concepts, and this section divides them according to the three major formulations of computing: logic, functions and procedures.

A Concept representation

The most important distinction in a KA framework is how systems represent what they learn. Knowledge representation has always been a central topic in artificial intelligence research, and Barr & Feigenbaum [12, pp.143-222] identify the representation schemes of ***logic, procedural representations, semantic networks, production systems, direct (analogical) representations, semantic primitives,*** and ***frames and scripts.*** It is hardly necessary to say that a data structure is not knowledge, any more than an encyclopaedia is. A representation of facts or rules only becomes knowledge when used by a program to behave in a knowledgeable way.

For both inductive and deductive concept learning the formalisms above suggest representing concepts in an appropriately powerful form of logic, such as predicate calculus. However, although formal logic provides a sufficient basis for deduction, as a foundation for induction it is at once too narrow and too powerful. It is too narrow because induction, based on search, is extraordinarily sensitive to the precise structure of the search space, and while different representations may be formally equivalent, they imply different search spaces. Extending the distinction that is often made between the epistemological adequacy of a representation (whether it is capable of expressing the facts that one has about the world) and its heuristic adequacy (whether the reasoning process actually gone through in solving a problem is expressible in the language) [45], one might characterize formal logic as epistemologically adequate but inductively inadequate for concept learning. On the other hand, logic is too powerful because the need to acquire knowledge automatically from teacher or environment and integrate it with what is already known means that only the simplest representations are used by programs for machine learning.

Any one representation will not encompass the broad application of concept learning techniques. Instead we use three: ***first order predicate calculus***, ***expressions composed of functions***, and ***procedures.*** These reflect fundamental formulations of computing which have been realized in logic, functional and imperative programming styles. Although equivalent in expressive power, the different representations are more or less appropriate for particular concept learning problems, depending on the nature of the examples, background knowledge, the way the complexity of concepts is measured, and the style of interaction with the teacher. For example, decision trees are naturally represented as logical expressions, polynomials as functions and robot tasks as procedures. Functional representations incorporate the powerful mathematics available for numbers. Procedures embody the notions of sequencing, side effects and determinism normally required in sequential, real world tasks.

Figure 2 shows how we classify concept representations in our framework. The abstract distinctions made are intended to reflect the nature of real concepts rather than just categorizing present concept learning systems. Figure 3 gives specific illustrations of concept and example representations. Numerical expressions constructed by composing arithmetic primitives comprise an important subclass of functions. Representations in propositional calculus are simpler and therefore easier to induce than those in predicate form.

There is an obvious overlap between logical and non-numerical function representations. For example, the concept of "appending" lists can equally well be written in logical and functional styles. The difference is that the logical form expresses a pure relation without distinguishing input and output, while the functional representation acts on the input lists to construct the output.[3] The two forms are shown in Figure 3. The framework includes both alternatives and leaves the choice to the user who can take into account the format of input examples and the way the concept will be applied.

1 Logic Many learning methods apply to examples that can be expressed as vectors of attributes in a form equivalent to ***propositional calculus***. This represents logical statements using predicates on constant terms, with connectives for disjunction, conjunction, negation and implication [51], as exemplified in Figure 3 by an attribute vector. The values an attribute can assume may be ***nominal***, ***linear***, or ***tree-structured*** [46]. A *nominal* attribute is one whose values form a set with no further structure; for example the set of primary colors. A *linear* attribute is one whose values are totally ordered; for example natural numbers. Ranges of values may be employed in descriptions. A ***tree-structured*** attribute is one whose values are ordered hierarchically. Only values associated with leaf nodes are observable in actual examples; concept descriptions, however, can employ internal node names where necessary. For example, the attribute ***shape*** might be hierarchical, having subsets such as ***polygon*** and *oval*.

First order ***predicate calculus*** allows variable terms in logical statements, and quantification over those variables [51]. Attribute vectors (propositional calculus) are not powerful enough to describe situations where each example comprises a "scene" containing several objects. The classic example is Winston's [70] ***arch*** concept, defined as three blocks, two having the attributes required of columns and the third having those required of a lintel, with the columns supporting the lintel and set apart from one another. Objects are characterized by their attributes. Moreover, pairwise (or more complex) relations may exist between them. This means that variables must be introduced to stand for objects in various relations. Such relations can be described by predicates which, like attributes, may be nominal, linear, or tree-structured. For example, a predicate **relative-position** might relate two objects with a pair of linear values **x-distance, y-distance**. Alternatively it could be tree-structured, with values such as **left-of, side-by-side, overlapping**. Objects and concepts are characterized by combinations of predicates.

2 Functions Typical functional expressions include many natural laws, as well as relationships between quantities sensed by a robot and parameters of a subsequent movement (eg see Figure 3). Functional representations are appropriate for nested and recursive numeric or non-numeric expressions. Any functional relationship $f(x)$ can be represented in logic as $\exists y \forall x,\ y = f(x)$, and this is no surprise since the two forms are expressively equivalent. But in a framework for induction, it is preferable to treat functional expressions separately and omit explicit quantifiers. An important difference is that functional representations of concepts must be single-valued, while logical expressions need not be; this greatly affects the search space involved.

A more suitable form of representation might be the lambda calculus, or some incarnation of it in pure LISP or other functional programming languages [23,30,55]. These are suitable representations in which to expressly prohibit aspects that distinguish functional from procedural representations in the framework, namely side effects (eg variable assignment) and reliance on sequential execution. Work on programming language semantics, also partly based on lambda calculus and often coupled with function programming languages (eg [23]) may suggest appropriate forms of representation. Existing function induction systems are specially designed for particular domains with little attention to more general forms.

3 Procedures Typical concepts in this category include robot procedures for assembly, welding, *etc*, and standard office procedures such as sorting mail [43,44,73]. The procedural formalism is suitable for representing sequential execution where side effects, such as variable assignment and real world outputs like robot movements, make it vital to execute the procedure in the correct order. To describe a procedural language formally, a binding environment model of execution is needed, instead of the simpler substitution model which suffices for pure functional representations [1]. Representations must be deterministic to be useful procedural concepts. Each of these decisions—order-dependence, determinism—have massive effects on the search space covered by an inductive search. Here also work on imperative programming language semantics (eg [25]) should provide ideas for simple, expressively adequate, consistent representations.

Note that the generalization of execution traces into a procedure is not reducible to an equivalent problem involving the generalization of non-sequential input-output pairs [11], as such a reduction will lose information about sequential changes in state.

B Example representation

Example representations are closely related to concept representations, for examples direct the search. Figure 2 classifies both in similar ways, while Figure 3 gives some illustrations. The differences between the two arise because examples are specific descriptions while concepts are general. In particular:

- Variables are permitted in concepts but not in examples. Thus for functions, examples are input-output tuples of constants; for relations they are tuples comprising either attribute values or relations on constants, and for procedures they are execution traces with no variables identified.
- If there are optional elements in the concept, only one option occurs in each example.
- Procedural control structures do not occur in examples, so example execution traces are single sequences without branches nor loops.

Examples of concepts which are couched in predicate calculus are always expressible in propositional calculus, as they contain no variables. Suppose the desired concept is:

$\texttt{color(x,c}_1\texttt{)} \wedge \texttt{color(y,c}_1\texttt{)} \wedge \texttt{darker(c}_1\texttt{,brown)} \wedge \texttt{larger(x,y)}$ (where x, y, $\texttt{c}_1$ are variables)

An example might be:

$\texttt{brick(obj}_1\texttt{)} \wedge \texttt{color(obj}_1\texttt{,yellow)} \wedge \texttt{color(obj}_2\texttt{,yellow)} \wedge \texttt{size(obj}_1\texttt{,2)} \wedge \texttt{size(obj}_2\texttt{,1)}$

IV BIAS

Search is fundamental to all concept learning. The search involved in both inductive and deductive learning is intractable unless it is strongly ***biased***, as the space of possibilities is too large. In effect, any search is biased towards discovering some things rather than others by the very way the space is constructed. It is particularly important in concept learning to recognize the in-built biases. Figure 1 shows two main categories.

A Background knowledge

How much knowledge does a system need before it can infer concepts from examples? There is an oft-quoted aphorism that in order to learn something, you must nearly know it already. The amount of knowledge already possessed about a problem domain critically affects the kind of things one can expect to be learned. Concept learning techniques can be classified as ***knowledge-sparse*** and ***knowledge-rich***, not so much on the basis of the amount of knowledge they embody as on the extent to which that knowledge is represented explicitly. Even the first kind embodies substantial prior knowledge in the form of languages in which examples and concepts are expressed. It is obviously desirable for a KA system to *learn* most of what it needs to know; otherwise its utility will be outweighed by the burden of expressing the prior knowledge. Unfortunately existing concept learning systems are able to learn very little, if any, of the background knowledge needed for them to work.[4]

Since background knowledge can be represented in diverse ways, it is hard to categorize its role in biasing search. The classification below, based on the amount of explicitly-represented background knowledge, is intended to be particularly relevant to KA applications. Although in some ways this is a continuous scale, in practice a different class of methods is used when substantial background knowledge is represented explicitly.

1 Knowledge-sparse techniques These methods are used when little or no domain knowledge is represented explicitly. There are three categories.

Parameter learning optimizes the numerical parameters of a pre-specified model. Such systems are often designed to detect regularities in noisy situations, and use knowledge in the form of statistical techniques. The structure of the model is not represented explicitly, and the system cannot modify it or reason about it. Generally, large numbers of examples are used to allow incremental, hill-climbing-style optimization and to overcome the effects of noise. Parameter learning is a marginal case of concept learning with a well-developed arsenal of statistical and other model-fitting techniques; we do not address it further here.

Similarity-based learning delineates the space of possible concept descriptions in advance, and searches it for ones which best characterize the structural similarities and/or differences between the examples presented. The space is often finite (but very large; too large for simple enumeration to be feasible). One key issue is whether learning proceeds incrementally, using each example to further refine a generalized concept description, or all-at-once, when the examples are batched together for processing. Another is whether an approximate or exact match of concept to examples is sought, in other words whether the examples may be contaminated by noise.

Hierarchical learning augments the description language as each concept is learned. This allows new descriptions to build upon old ones; so that background knowledge grows. Such learners are open-ended in that concepts of arbitrary complexity may be constructed. Search is controlled by apposite choice of order in which concepts are learned; thus the teacher plays a crucial role in directing learning.

2 Knowledge-rich learning These techniques employ considerable knowledge about the target domain. They work by relating new information to the existing body of knowledge. We distinguish two categories.

Explanation-based learning assimilates a substantial chunk of information by intensive analysis of just one example. Deductive techniques are used to operationalize that part of the knowledge base which relates to the example. Such a system does not learn "new" knowledge, but articulates rules which are already implied by its knowledge base. To do this it must explain—in effect, prove—the example given; and then generalize the explanation to account for other examples. A key issue is where the explanation comes from in the first place; from the system (following a very extensive search), from the teacher, or from a teacher-directed search.

Learning by discovery is when systems operate autonomously, performing experiments to enhance their knowledge base. They can proceed either deductively or inductively. They search for "interesting" things, forming conjectures by examining a few examples of concepts and noticing coincidences; perhaps attempting to prove these conjectures. One key issue is the mechanism by which interesting conjectures are singled out and others discarded. Another is the distribution of resources between competing lines of development. It is not yet clear whether such open-ended search is capable of being controlled usefully.

B Constraining induction

Concept learning is rendered tractable by constraining the search to exclude major portions of the potential space. In existing systems the designer does this by ruling out possible forms of a concept. Ideally, a mechanism would be provided whereby a KA system could learn about the constraints which operate in a particular domain, but the general problem of learning constraints has yet to be tackled.[5] Note that it is not amenable to a purely formal treatment, for biasing constraints can only be expressed in logic as properties of predicates, so reasoning about bias entails the use of higher-order logics (which are undecidable). This section examines three different ways in which induction can be constrained.

Conceptual bias limits the vocabulary for expressing concepts. It applies equally well to predicate calculus, functional, and procedural representations. The vocabulary in question is not the entire set of symbols from which concepts are constructed, but only those that appear in examples. In logical representations of playing-card concepts, for instance, values might be represented by elements of the set {2,3,4,5,6,7,8,10,J,K,Q,A}, or just by the descriptors {**face-card, non-face-card**}. However, conceptual bias does not restrict the use of non-domain symbols such as $\wedge$, $\vee$ and $\Longrightarrow$.

In functional representation, conceptual bias determines what are the inputs (variables) and outputs (results) of expressions. For example, when learning functional expressions embedded in robot arm procedures, inputs might be robot joint angles, or the hand position in world coordinates. In procedural representations conceptual bias determines which domain objects can be referenced, in other words what the variables can be. For example, robot procedures might be able to refer to a variable giving the angle of contact between the arm and an obstacle, or alternatively to variables which record the touch pattern and forces at the points of contact.

A search can be conceptually biased to the extent that the system designer knows what components are needed in concepts. Conceptual bias is implicitly present in any formulation of a problem. For instance, the fact that physical attributes like **dog-eared** are excluded in a playing-card domain is a form of bias.

Composition bias constrains how the allowed vocabulary can be used to construct concepts. It generalizes the notion of *logical bias* [22] to functional and procedural representations as well as logical ones. It limits the connectives used when forming expressions from vocabulary terms, thereby restricting the language in which concepts may be couched. For example, disjunctions might be prohibited so that only conjunctive descriptions such as Spade $\wedge$ Face-card are permitted. At any rate, unconstrained disjunctive combinations should be discouraged, to stop a concept degenerating into a list of all examples seen.

In functional representations composition bias determines the vocabulary of built-in functions and operators, and the syntax by which concepts may be constructed. For example, in searching for a functional expression we might consider combining terms with operators from the set $\{\times, \div, +, -, \circ\}$ (where $\circ$ is function composition). In both functional and procedural representations, compositional constraints can be used to govern how terms match, ensuring that data types and physical quantities are combined appropriately. Composition bias in procedural representations is reflected in the use of a limited repertoire of control constructs, like **if**...**then**...**else**... and **while**...**do** ...; and even in the presupposition of sequential execution. In future there may be greater use of parallel procedural representations, where a construct like **sequence** is necessary to indicate sequentiality explicitly. Such constraints reflect at a rather deep level the style of computation envisaged.

There is an interesting relationship between conceptual and composition bias, and formal grammars. A grammar has a terminal vocabulary, the set of all symbols in the language it generates, and a set of production rules for sentence formation. To any concept space there corresponds a grammar which generates possible concepts. Conceptual biasing determines a vocabulary which is a *subset* of that grammar's terminal symbol vocabulary. It specifies only *content* symbols (those that refer to things in the domain) and not the non-content symbols such as logical connectives. Composition bias determines both the allowable non-content symbols *and* the production rules that may be used to form concepts.

For example, consider a grammar which generates concepts in the playing-card domain like Spade $\wedge$ Face-card. Spade, Face-card, and $\wedge$ are terminal symbols for the grammar. Spade and Face-card are content symbols belonging to the vocabulary determined by conceptual bias. $\wedge$ and a production rule allowing it to combine as an infix operator are both determined by composition bias. Spade and Face-card are aspects of the domain, whereas $\wedge$ and the production rule are aspects of concept composition.

Preference ordering. Creative science proposes hypotheses that explain observations. This parallels our much narrower notion of concept learning, with the notable difference that scientific hypotheses are arrived at by sometimes amazing intuitive leaps, rather than constrained searches. Any observations could be explained by an indefinite number of hypotheses, and scientists apply all kinds of intellectual skills to help them come up with appropriate ones. Hypotheses cannot be justified deductively, for they are induced from observed examples. There is no formal way to choose between competing ones, so long as they explain the examples given and can successfully predict crucial examples not in the original set [56].

More informally, hypotheses are preferred if they are elegant, simple, parsimonious, *etc.* These are patently subjective qualities, hard to capture formally. KA systems must often choose between competing concepts but cannot afford the luxury of ill-defined measures. Methods for determining a preference order inevitably depend on the syntax of the language used to express concepts.

A straightforward approach is to apply Occam's razor by measuring the brevity of concept descriptions. One concept is simpler than another if it is shorter when written down. Length can be measured in various ways, including

- number of disjuncts in logical expressions
- number of function calls in functional expressions
- number of function calls plus total number of function arguments
- length of function expressions represented as character strings
- number of loops and conditionals in procedures
- number of statements in procedures
- number of states or transitions in a state-diagram representation [21].

A different approach, model-maximization [22], uses the extension rather than the intension of a concept. One concept is preferred over another if its extension is a superset of the other's. The KA system must know how to order concepts according to their extensions; the ordering will be partial as extensions may overlap. For example the concept `Spade` $\wedge$ `Face-card` would be preferred to `Spade` $\wedge$ `Face-card` $\wedge$ `Room-temperature(30 degrees)` as it includes all of the extension of the latter concept, and more. Neither would be ordered with respect to `Black` $\wedge$ `Queen`.

V TEACHER INTERACTION

A concept learning system's practical utility obviously depends critically on its teaching requirements. Teaching differs from programming in that a teacher does not use a formal model of the learner, whereas a programmer normally expects to know exactly how his instructions are going to be interpreted. In other words, a teacher adopts the *intentional stance* while a programmer adopts the *design stance* [19,20]. Ideally, therefore, to be a good teacher one need know nothing about the internal structure of the learner, nor about the representation it uses for concepts.

Moreover, experts who acquire their knowledge from example cases are unlikely to have a clear analytical understanding of the task domain; otherwise more explicit methods of knowledge transfer will probably prove more appropriate. In general, people find it difficult to translate their own expertise into explicit descriptions. Consequently concept learning systems should be able to work with the kind of examples that a teacher finds it natural to provide, ordered in a way that is natural to him.

As Figure 1 shows, the framework distinguishes two important aspects of the teacher's interaction with the KA system, discussed below along with a third aspect. The teaching requirements and conditions under which learning is to occur form an essential part of our framework. Anything that is learned must be communicated through the expert- or user-machine interface.

A Presentation: are examples presented incrementally or in batches?

It may be that the current description must be augmented as each example is encountered; alternatively it might be possible to process all examples together. Batch operation is infeasible for systems which attempt sustained learning. We might expect incremental learning to be faster, as the contribution from each example is made available sooner. An important question for a teacher is: In the incremental case, is the final concept supposed to be independent of the order of examples?

B Examples: what examples are provided?

Examples are the main—often the only—specific information a system has to help it identify a particular concept, and the choice of examples is crucial. One important practical question is: Can the system generate its own examples, ones important to its own concept search strategy? A learner has the opportunity to design test cases for specific hypotheses about the target concept, whereas teacher-provided examples may be irrelevant. A second practical question is: How reliable are examples? They may contain noisy components, or be occasionally misclassified. Most concept learning systems assume exact data so that structural regularities can be learned without the problems of statistical detection.

An important question is whether negative examples are provided as well as positive ones. There is a basic theoretical distinction between presenting positive examples only, and presenting both positive and negative examples (or, equivalently, allowing the learning system to choose examples itself and have them classified by an informant). It can be shown formally that primitive recursive languages (which include the context sensitive, context free, and regular languages) can be learned eventually with the second and third methods, whereas for positive presentations only, the inclusion of even one infinite language in the set of potential concepts can destroy learnability [24,72]. An additional, curious, theoretical result is that descriptions unlearnable from positive and negative examples *can* be learned if the presentation sequence is sufficiently regular [24].

C How good is the teacher?

This third area concerns the co-operation and pedagogical skills of the teacher. Learning situations range from having no teacher at all (input coming directly from the environment) through a naive user, a domain expert, a skilled trained teacher, to a "teacher" who is prepared to program in a conventional programming language and thereby circumvent the need for learning. Must the teacher show all working, give "near-miss" examples, teach simple concepts before complex ones, and so on [71,72]?

A skilled teacher will select illuminating examples himself and thereby simplify the learner's task. The benefits of carefully constructed examples were appreciated in the earliest research efforts in concept learning [59,70]. The notion of a sympathetic teacher has been formalized in terms of "felicity conditions", constraints imposed on or satisfied by a teacher that make learning better than from random examples [68]. The teacher should classify examples correctly, point out absences explicitly, show all work, avoid glossing over intermediate results, and introduce only one essentially new feature per lesson. Examples should not by coincidence include features that might mislead the learner [71,72]. Although it does not increase formal learning power, the possibility of a system constructing its own examples and having them classified by an informant has considerable potential to speed up learning and reduce dependence on the skill of the teacher.

VI USING THE FRAMEWORK

It is not possible, in the present state of the art, to supply a comprehensive set of rules to determine which concept learning scheme to use, given desired concept and example representations, background knowledge available, form of teacher interaction, and appropriate biases for concept representations. This section indicates how to go about making suitable choices by providing examples of particular kinds of system. Table 1 summarizes how the framework can be used to select from presently available systems.

Faced with a practical KA problem, the first decisions to make are how to represent concepts and examples. The two are closely related. Suitable forms of concept representation will be dictated by the requirements of the knowledge-based system and the kind of examples available. Sometimes the example representation will dominate the decision, while in other situations the desired concept format will force examples into a particular mould.

The next four sections discuss logic-based concept representation, differentiated by the various ways that background knowledge can bias the search: similarity-based, hierarchical, explanation-based and discovery-based (see Figure 1). Logical representations are indicated by the predominance of logical relationships in examples or concepts. The possibility of using attribute values strongly suggests the simpler propositional calculus representation. Similarity-based system methods are appropriate when many examples are available, or when it is not possible to define a domain theory in advance. If concepts must be built on earlier learned ones, a hierarchical method is indicated. If a domain theory is known then an explanation- or discovery-based system can utilize it.

The final two sections deal with functional and procedural concept representations. The former should be chosen when functional relationships are desired; the latter when sequential tasks are acquired.

A Logical representations: similarity based

Given a set of objects which represent examples and counterexamples of a concept, a similarity-based learner attempts to induce a generalized description that encompasses all the examples and none of the counterexamples. Interesting general issues include the conditions under which the procedure converges to a single description, whether the system can know that it has converged, whether the final concept may depend on the order of presentation of examples, and whether the training set is expected to be exhaustive (induction is merely summative) or representative ("generalization" is expected).

Version space The "version space" approach to concept learning [48] transforms the inductive problem of generalization into a deductive one by circumscribing the way in which descriptions are expressed and searching for ones which fit the examples given. It postulates a language in which objects are expressed, a language in which concepts are expressed, a predicate which determines whether a particular object matches a particular description, and a partial ordering on descriptions (interpreted as generalization/specialization). Given a set of positive and negative examples of a target description, a simple search algorithm exists which finds all descriptions that are consistent with the examples. This set is called the "version space".

The method applies simply and directly when each object is described by a fixed set of properties, usually represented as an attribute vector, which is equivalent to a description in propositional logic. Its performance in such domains has been studied extensively. Allowing disjunctions in the description language causes the version space to explode; while even with purely conjunctive concepts, one version space boundary can grow exponentially in the number of examples. This and other complexity results are presented by Haussler [27].

In structural domains, each example (or counterexample) comprises a "scene" containing several objects, expressed in predicate logic. Part of the problem in matching a scene with a structural description is determining an appropriate mapping between objects in the scene and those specified in the description. This mapping will have different interpretations depending on whether the scene is to comprise or merely to contain the desired object [66]. Several theoretical results indicate that, even in the simplest cases, extreme computational complexity can be involved in working with version spaces of structured objects [28].

Creating decision rules A decision tree or collection of rules may be constructed which discriminates positive from negative examples. In contrast to version space methods, it is assumed that all examples are available and can be processed together.

In the case of a tree, the root specifies an attribute to be selected and tested first, then depending on its value subordinate nodes dictate tests on further attributes. The leaves are marked to show the classifications of the objects they represent. For two-class problems these are simply "positive" and "negative", but it is easy to distinguish more than two classes. The ID3 algorithm [57] uses an information-theoretic heuristic to find a simple tree which classifies all examples given. This procedure has been studied extensively and adopted for some commercial applications. When presented with noisy data, it constructs huge decision trees which reflect the detail of every example seen. The resulting uneconomical representation is not a suitable vehicle for generalization, and several research efforts are underway to develop improved algorithms which cope with noise [50,58].

In the case of production rules, the training set is used to construct a set of rules which can be interpreted by an expert system in standard forward- or backward-chaining manner. While any decision tree can easily be converted to rules, the rules may contain redundancies which can be eliminated by generating them directly from the examples [16].

B Logical representations: hierarchical learning

It would be attractive if learning systems could build upon concepts already learned, and use them as components in newly-constructed descriptions. This might allow learning to be sustained over an extended period of time, instead of being done on a one-off basis.

Concept trees A system called MARVIN learns hierarchical structures of concepts [60,61]. Given an example, it searches for ways to express it in terms of known concepts. Instead of awaiting further examples from the teacher to constrain its choice of expressions, however, it selects a tentative description, synthesizes a "crucial object" to determine if that description is a correct generalization, and asks whether it too is an example of the concept. If so, the tentative description is accepted and attempts are made to generalize it further. If not, the tentative description is specialized to rule out the negative example and a new crucial example is synthesized. Thus a sequence of generalizations and specializations is made, each being tested by asking the teacher to classify a crucial example. When all possibilities for generalization have been exhausted, the description is stored and the teacher is asked for another example of the concept.

Semantic nets The method implemented by MARVIN breaks down if the concepts are not structured and taught in the form of a tree. An extreme case of non-hierarchical representation is where each object is represented by an attribute vector. Real-world knowledge bases are likely to fall between the extremes of hierarchical and completely flat representations. MARVIN has been extended to deal properly with such situations by thoroughly investigating competing hypotheses at each stage. If a crucial object exists within the current tentative hypothesis but not within any competing ones, it is presented to the teacher. If such an object does not exist, the alternative hypotheses are investigated to discover the most general valid description [34].

C Logical representations: explanation-based

It is often argued that an explanation-driven process is essential for reasonable generalization. Correlations derived purely from empirical observations are much less convincing than a theory which explains them. The earliest example of explanation-based learning is LEX-II, which learns problem-solving heuristics in the domain of symbolic integration. More recently, the method has been clearly illustrated in a blocks-world application [49], and several extensions to the idea have been identified [18]. It usually presumes the existence of a strong domain theory in which proofs can be constructed that show why a particular example is valid, but some experiments have been reported with a kind of explanation-based learning supported by a very weak theory in the realm of physical causality for everyday events [54].

D Logical representations: discovery-based learning

Isolated attempts have been made to meet what is perhaps the major shortcoming of similarity-based learning, that examples are lifted out of the world, cleaned up, and presented to the system by a teacher who makes all the important decisions about when to create new concepts. Still the most impressive are the AM and EURISKO systems of Lenat (see [38,39] and in particular [40]) which work on their own to discover interesting concepts in elementary mathematics and interesting heuristics for further discovery. However, research in this area is quite immature and does not yet form a foundation for practical KA techniques.

E Functional representations

Representing concepts as functions is appropriate when examples take the form of input-output pairs, and the knowledge to be acquired is an expression for computing the output of each pair from its input. For numerical functions, polynomial approximation can be used. However, this is not as useful as it appears at first sight. For one thing, the issue of complexity arises again. While from some points of view a transcendental function like $\sin x$ may be adequately approximated by a high-order polynomial, there are usually advantages in a more concise description.

MARVIN, discussed above, can be used for inducing logical relationships akin to functions. Examples of concepts it has learned include list manipulation, arithmetic, and simple insertion sort.

NODDY Function induction is employed as a component of NODDY, a system for learning robot programs from traces of their execution [9,10]. When presented with a list of input-output pairs gleaned from similar stages of successive incarnations of the same robot procedure, and a list of known constants that can be used, it attempts to induce a single function for all pairs. If found this function becomes part of the emerging procedure as it captures a functional relationship in each example trace. Strong composition bias, in the form of type matching, is built in to the algorithm to reduce the search space.

BACON One AI system which works on wholly numeric functions is BACON, which attempts to discover empirical scientific laws by inducing functions that account for observed data [35]. It is given as input a table of values of dependent and independent variables, and seeks a functional expression using intermediate *theoretical* subexpressions to direct the search [53]. Recent work is directed at making it able to deal with noisy data using a hill-climbing strategy [37].

COPER A more structured approach to deducing physical laws is to use dimensional analysis in concert with observed data. COPER works with physical quantities and uses dimensional analysis to tell whether the independent variables are sufficient to determine the dependent variable's value uniquely [33]. It selects an appropriate dimensional base and uses it to generate a polynomial expression.

Synthesizing LISP functions The synthesis of LISP functions from a few example input-output pairs of list structures has been extensively studied [65]. Quite restrictive assumptions are placed on the examples. All atoms in the input list structure must be unique; every atom that occurs in the output must also occur in the input; and there must be a total order relation corresponding to "simplicity" on the inputs. Traces of possible functions are formed directly from the input-output pairs and used to create a suitable recursive function. Simple but powerful forms for possible functions are assumed, comprising an overall LISP conditional with function calls and basic list primitives forming the branches of the conditional. Recurrence relations among the input-output pairs enable recursive calls to be induced.

F Procedural representations

Concepts are best represented as procedures when the examples are sequential traces of the actions required to perform some task. Like other inductive problems, acquisition of procedures from examples is intractable in general because of the huge spaces involved. Practical systems must implement methods for limiting the search based on knowledge of the domain.

NODDY acquires robot procedures, complete with control information which is not explicitly present in the examples [9,10]. It copes with problems of action sequencing, and also handles real numbers representing angles and distances. It employs an explicit, pre-programmed, generalization hierarchy, and pre-programmed information on about 30 basic mathematical and set-theoretic operators that may be combined to create complex generalizations. Examples are traces of the desired procedure. The first trace is taken to be the initial version of the procedure. As further traces are seen they are merged with the nascent procedure, generalizing it in various ways. The system cannot reconsider generalizations it has made in the current version of the procedure, and therefore adopts a conservative policy of requiring considerable evidence before generalizing.

VII CONCLUSION

This paper has developed an integrative framework for describing concept learning techniques which enables their relevance to particular knowledge engineering problems to be evaluated. It provides a general basis for thinking about concept learning in knowledge acquisition problems, techniques and applications, and forms a starting point for the future development of detailed formal design rules. Distinctions are made for the concept and example representation, the search bias, and the mode of teacher interaction.

Concept learning has suffered its share of what Parnas [52] refers to sarcastically as the "promise of Artificial Intelligence"; that AI is not so much a field of great promise but that AI researchers *make* great promises, mostly unfulfilled ones. Despite the heady aura of optimism which pervades many research papers on the subject, it is difficult to identify specific cases of concept learning systems actually doing useful things in the service of people. Sober reflection reveals that this is likely because learning, as the touchstone of intelligence, is *hard*; we can confidently expect progress in machine learning to be slow, erratic, and painful.

Nevertheless there are techniques of learning that are certainly ripe for exploitation in knowledge acquisition systems. Similarity-based methods such as version space and especially ID3 encapsulate neat algorithms that can be used to abstract properties common to large sets of examples. Hierarchical methods may not be far behind, although at present they rely unduly on the skill of the teacher and lack the robustness which is essential in a system intended for practical use. Knowledge-rich learning must wait to see how the field of expert systems develops, for it presupposes deep structural models of the domain and not simply rules of thumb which simulate expert performance at a superficial level. Function induction and the synthesis of procedures from traces are promising new technologies that have yet to move outside research laboratories.

What is certain to be required, however the field develops, is an integrative framework which permits different mechanisms to be compared and contrasted. It is essential to be able to describe and categorize learning techniques in a way that makes them readily accessible to potential users. It is also necessary, in view of the present state of turmoil in the machine learning field, to provide researchers with common ground so that they can evaluate different approaches. There is pressing need for more formalism and more attention to the bias, often unstated, which is concealed deep within the bowels of systems for machine learning.

This paper has developed a framework for describing concept learning techniques which enables their relevance to knowledge engineering problems to be assessed. No doubt it will evolve and grow as we gain more experience with it, and as new techniques emerge.

REFERENCES

[1] Abelson, H. and Sussman, G.J. with Sussman, J. *Structure and Interpretation of Computer Programs.* MIT Press.

[2] Andreae, J.H. *Thinking with the Teachable Machine.* Academic Press, 1977.

[3] Andreae, J.H. Man-Machine Studies. Progress Reports nos. UC-DSE/1-29 to the Defence Scientific Establishment. Dept of Electrical and Electronic Engineering, University of Canterbury, Christchurch, New Zealand. (These reports are also available from the N.T.I.S., 5285 Port Royal Rd, Springfield, Virginia 22161.) 1972-1987.

[4] Andreae, J.H. Purposeful Computer Systems. Proc.Int.Conf. on Future Advances in Computing, Christchurch, N.Z. Published by Dept of Computer Science, University of Calgary, Alberta 2TN 1N4, Canada,1-26, 1986.

[5] Andreae, J.H. and Andreae, P.M. Machine learning with a multiple context. Proc.9th IEEE Int.Conf.on Cybernetics and Society, Denver, October, 734-9, 1979.

[6] Andreae, P.M. and Andreae, J.H. A teachable machine in the real world. *Int.J.of Man-Machine Studies, 10,* 301-12, 1978.

[7] Andreae, J.H. and Cleary, J.G. A New Mechanism for a Brain. *Int.J. Man-Machine Studies, 8,* 89-119, 1976.

[8] Andreae,J.H. and MacDonald,B.A. Expert control for a robot body. Research Report No. 87/286/34, Department of Computer Science, University of Calgary, 1987.

[9] Andreae, P.M. Constraint limited generalization: acquiring procedures from examples. Proc American Association on Artificial Intelligence, Austin, TX, August, 1984.

[10] Andreae, P.M. Justified generalization: acquiring procedures from examples. PhD Thesis, Department of Electrical Engineering and Computer Science, MIT, 1984.

[11] Angluin, D. and Smith, C.H. Inductive Inference: Theory and Methods. *Computing Surveys, 15*(3) 237-269, September, 1983.

[12] Barr, A. and Feigenbaum, E.A. (Editors) *The handbook of artificial intelligence Volume II.* HeurisTech Press, Stanford, CA, 1982.

[13] Boose, J. and Bradshaw, J.M. Expertise transfer and complex problems. *Int J Man-Machine Studies, 26*(1) 3-28, January, 1987.

[14] Bruner, J.S., Goodnow, J.J., and Austin, G.A. *A study of thinking.* Wiley, New York, 1956.

[15] Carter, C. and Catlett, J. Assessing credit card applications using machine learning. *IEEE Expert, 2*(3) 71-79, Fall, 1987.

[16] Cendrowska, J. PRISM: an algorithm for inducing modular rules. *Int.J. Man-Machine Studies,* in press.

[17] *The concise oxford dictionary.* Sixth edition. Oxford University Press, 1976.

[18] DeJong, G. and Mooney, R. Explanation-based learning: an alternative view. *Machine Learning, 1*(2) 145-176, 1986.

[19] Dennett, D.C. *Brainstorms.* Harvester Press, Brighton, Sussex, 1981.

[20] Dennett, D.C. *The intentional stance.* MIT Press, 1987.

[21] Gaines, B.R. System identification, approximation and complexity. *Int J General Systems, 3,* 145-174, 1977.

[22] Genesereth, M.R. and Nilsson, N.J. *Logical foundations of artificial intelligence.* Morgan Kauffman, 1987.

[23] Glasser, H., Hankin, C. and Till, D. *Principles of functional programming.* Prentice-Hall, 1984.

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

[25] Gordon, M.J.C. *The denotational description of programming languages.* Springer-Verlag, 1979.

[26] Haack, S. *Philisophy of logics.* Cambridge University Press, Cambridge, England.

[27] Haussler, D. Bias, version spaces, and Valiant's learning framework. Proc 4th International Workshop on Machine Learning, 324-336, Irvine, CA, June, 1987.

[28] Haussler, D. Learning conjunctive concepts in structural domains. Proc AAAI, 466-470, 1987.

[29] Hawkins, D. An analysis of expert thinking. *Int J Man-Machine Studies, 18*(1) 1-47, January, 1983.

[30] Henderson, P. *Function programming.* Prentice-Hall, 1980

[31] Hunt, E.B. *Concept learning: an information processing problem.* Wiley, New York, 1962.

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

[33] Kokar, M.M. Determining arguments of invariant functions. *Machine Learning, 1*(4) 403-422, 1986.

[34] Krawchuk, B.J. and Witten, I.H. On asking the right questions. Proc 5th Intl Machine Learning Conference, Ann Arbor, Michigan, June, 1988.

[35] Langley, P., Bradshaw, G.L., and Simon, H.A. Rediscovering chemistry with the BACON system. In *Machine learning,* Volume 1, edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 307-329. Tioga, Palo Alto, CA, 1983.

[36] Langley, P., Simon, H.A., Bradshaw, G.L. and Zytkow, J.M. *Scientific discovery—computational explorations of the creative process.* MIT Press, Cambridge, MA, 1987.

[37] Langley, P., Zytkow, J.M., Simon, H.A., and Bradshaw, G.L. Rediscovering chemistry with the BACON system. In *Machine learning,* Volume 2, edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 425- 469. Morgan Kaufmann, Los Altos, CA, 1986.

[38] Lenat, D.B. The ubiquity of discovery. *Artificial Intelligence, 9,* 257-285, 1978.

[39] Lenat, D.B. EURISKO: a program that learns new heuristics and domain concepts. *Artificial Intelligence, 21,* 61-98, 1983.

[40] Lenat, D.B. and Brown, J.S. Why AM and EURISKO appear to work. *Artificial Intelligence, 23,* 269-294, 1984.

[41] MacDonald, B.A. Designing Teachable Robots. PhD thesis, University of Canterbury, Christchurch, New Zealand, 1984.

[42] MacDonald,B.A. Improved robot design. *Trans.IPENZ Elec/Mech/Chem section, 14,*(1/EMCh), 33-48, 1987.

[43] MacDonald, B.A. Programming computer controllers by giving examples. Proc. IEEE Wescanex. 1st Conf. on Programmable Control Systems, Edmonton, Sept 9-10, 49-53, 1987.

[44] MacDonald, B.A. and Witten, I.H. Programming Computer Controlled Systems by Non-experts. Proc IEEE Systems, Man and Cybernetics Annual conf, Oct 20-23, Alexandria, Virginia, pp.432-7, 1987.

[45] McCarthy, J. and Hayes, P.J. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4,* edited by Meltzer, B. and Michie, D., pp 463-502. Edinburgh University Press, 1969.

[46] Michalski, R.S. A theory and methodology of inductive learning. In *Machine learning,* Volume 1, edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 83-134. Tioga, Palo Alto, CA, 1983.

[47] Michalski, R.S. and Chilausky, R.L. Learning by being told and learning from examples: an experimental comparison of two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *Int. J. Policy Analysis and Information Systems, 4*(4), 1980.

[48] Mitchell, T.M. Generalization as search. *Artificial Intelligence, 18,* 203-206, 1982.

[49] Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. Explanation-based generalization: a unifying view. *Machine Learning, 1* (1) 47-80, 1986.

[50] Niblett, T. and Bratko, I. Learning decision rules in noisy domains. in *Research and development in expert systems III,* edited by M.A.Bramer, pp 25-34. Cambridge University Press, Cambridge, England, 1986.

[51] Nilsson, N.J. *Principles of artificial intelligence.* Tioga, 1980.

[52] Parnas, D.L. Why engineers should not use artificial intelligence. Proc CIPS Edmonton, 39-42, Edmonton, Alberta, Canada, October 1987.

[53] Pauli, D and MacDonald, B.A. Inducing functions in robot domains. Research Report No. 88/296/08, Computer Science department, University of Calgary.

[54] Pazzani, M.J. Inducing causal and social theories: a prerequisite for explanation-based learning. Proc 4th International Workshop on Machine Learning, 230-241, Irvine, CA, June, 1987.

[55] Peyton-Jones, S.L. *The implementation of functional programming languages.* Prentice-Hall, 1987.

[56] Popper, K.R. *The logic of scientific discovery.* Hutchinson, London, 1968.

[57] Quinlan, J.R. Induction of decision trees. *Machine Learning, 1*(1) 81-106, 1986.

[58] Quinlan, J.R. The effect of noise on concept learning. In *Machine learning,* Volume 2, edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 149-166. Morgan Kaufmann Inc, Los Altos, C, 1986.

[59] Rissland, E.A. The problem of intelligent example selection. To be published in *Int.J. Man-Machine Studies,* 1988.

[60] Sammut, C. and Banerji, R. Hierarchical memories: an aid to concept learning. Proc International Machine Learning Workshop, 74-80, Allerton House, Monticello, IL, June 22-24, 1983.

[61] Sammut, C. and Banerji, R. Learning concepts by asking questions. in *Machine learning,* Volume 2, edited by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, pp 167-191. Morgan Kaufmann Inc, Los Altos, CA, 1986.

[62] Schank, R.C., Collins, G.C. & Hunter, L.E. Transcending inductive category formation in learning. *Behavioral and Brain Sciences, 9*(4) 639-651, December, 1986.

[63] Shalin, V.L., Wisniewski, E.J., and Levi, K.R. A formal analysis of machine learning systems for knowledge acquisition. To be published in *Int J Man-Machine Studies,* 1988.

[64] Shapiro, E.Y. *Algorithmic program debugging.* MIT Press, Cambridge, MA, 1983.

[65] Smith, D.R. The synthesis of LISP programs from examples: a survey. In *Automatic Program Construction Techniques* edited by A.W. Biermann, G. Guiho and Y.Kodratoff, Macmillan, 1984.

[66] Stepp, R.E. Machine learning from structured objects. Proc 4th International Workshop on Machine Learning, 353-363, Irvine, CA, June, 1987.

[67] Thimbleby, H. *User interface design—a handbook for thinkers.* Addison-Wesley, in press.

[68] Van Lehn, K. Felicity conditions for human skill acquisition: validating an AI-based theory. Research Report CIS-21, Xerox PARC, Palo Alto, November, 1983.

[69] Williams, H.C. Factoring on a computer. *Mathematical Intelligence, 6*(3), 29-36, 1984.

[70] Winston, P.H. Learning structural descriptions from examples. in *The psychology of computer vision*, edited by P.H.Winston. McGraw Hill, New York, NY, 1975.

[71] Witten, I.H. and MacDonald, B.A. Concept learning: a practical tool for knowledge acquisition? Proc 7th Intl Workshop on expert systems and their applications, Avignon, France, 1535-55, May, 1987. Also available as Research Report 87/253/01, Department of Computer Science, University of Calgary.

[72] Witten, I.H. and MacDonald, B.A. Using Concept Learning for Knowledge Acquisition. *Int.J.Man-Machine Studies.* Also presented at *AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* Banff, Canada, October, in press.

[73] Witten, I.H., MacDonald, B.A. and Greenberg, S. Specifying Procedures to Office Systems. Proc. Conf. on Automating Systems Development, April, Leicester, England, 1987.

[74] Wittgenstein, L. *Philosophical investigations.* Blackwell, Oxford, UK, 1958.

[75] Woods, W.A. Transition network grammars for natural language analysis. *Communications of the ACM, 13*(10) 591-606, October, 1970.

Figure 1: The important distinctions made in the framework.

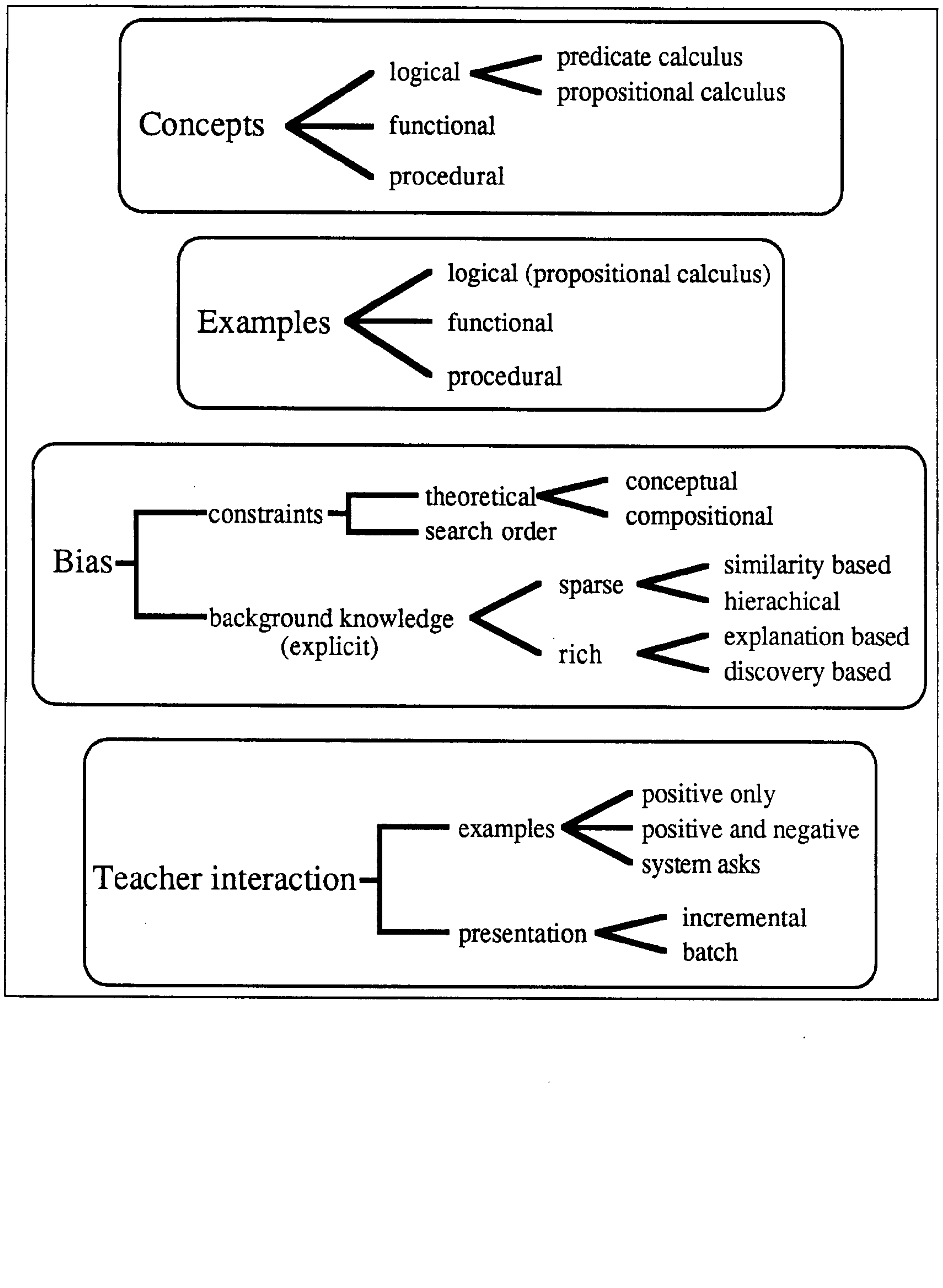


Figure 2: Framework for representing concepts (top) and input examples (bottom).

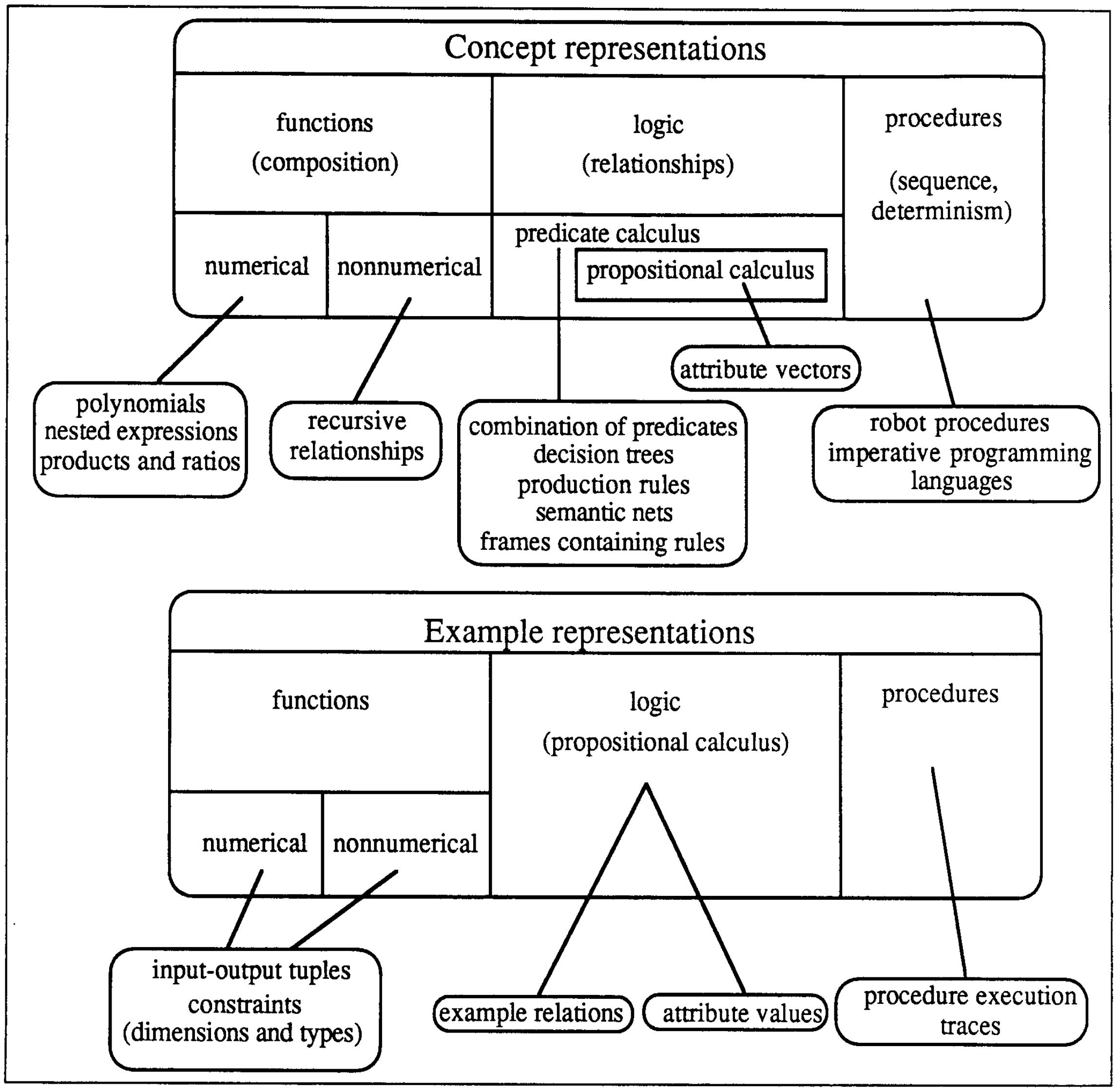
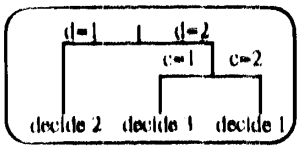


Figure 4. Concept and example representations, together with some samples

	Concepts	Examples	
attribute vector	(color= ?, shape = circle)	(color = red, size = 2, shape = circle)	attribute values
predicate combination	(shape(y)=convex, relative position(x,y) = inside)	inside(x,y), shape(x)=circle, shape(y)=square, color(y)=red,y =object1, color(x)=blue,x=object2	relation tuple
recursive logical relationship	list-append(L1,L2,L) is true if it is ([],L,L) or it is ([Lhead|L1tail],L2,[Lhead|Ltail]) and list-append(L1tail,L2,Ltail) is true	([a],[b],[a,b])	relation tuple

decision tree	 d=1
	d=2
	c=1
	c=2
	decide 2
	decide 1
	decide 1
		a=2, b=2, c=3, d=1 --> 2
		a=3, b=1, c=5, d=1 --> 2
		a=2, b=2, c=1, d=2 --> 1
		a=5, b=4, c=1, d=2 --> 1
			attribute values

production rules	d=1 --> decide 2
	d=2, c=1 --> decide 1
	d=2, c=2 --> decide 3		

recursive functional relationship	L=list-append(L1,L2): L is L2 if L1 is [] else L is [Lhead|Ltail], where Lhead is L1head and Ltail is list-append(L1tail,L2)	([a],[b] --> [a,b])	non numeric input output tuple
polynomial expression	$O = I1 \times I2 + 5 \times I2^2$	(6,2) --> 32 (9,3) --> 72 all integers	numeric input output tuple

nested numeric expression	position(x-part(input_1)+xpart(input_2), y-part(input_1))	position(1,4) position(2,6) --> position(3,4)
		position(3,3) position(1,7) --> position(1,7)
			numeric input output tuple

directed graph procedure representation	if contact @x then move x@y move 2x@y+15	contact @ 1 move 1@30 move 2@45	execution trace

Table 1: Representation, bias and interaction used (abbreviations are from Figure 1).

	Representation		Bias	Interaction			
	concept	example	constraints	knowledge	examples	presentation	sample systems
version space	attributes or predicates	attribute-values relation tuples	predetermined concept language	SB	±	I	many [27,28,48,66]
decision rules	decision tree production rules	attribute-values	conjunction of attribute values	SB	±	B	ID3 [50,57,58] PRISM [16]
concept trees semantic nets	recursive logical relationship	nested relation tuples	disjunction of conjunctions	H	S	I	MARVIN [60,61] ALVIN [34]
integration rules	operationalized rules	sample integration	predetermined concept language	EB	+	I	LEX-II
blocks world	predicate logic rules	sample predicate	predetermined concept language	EB	+	I	[49]
physical causality	frames containing rules	structure of frames	predetermined concept language	EB	+	I	[54]
mathematical laws	frames containing rules	domain simulation	heuristics for "interestingness"	DB	±	-	AM, Eurisko [38,39,40]
robot functions	nested expression	IO tuples	restricted constants and arguments	rich	+	B	NODDY [9,10]
scientific laws	composition of linear and ratio operators polynomial of transformed arguments	IO tuples IO tuples + dimensions	combination of ratios and products polynomial on dimensional base	rich	+	B	BACON [35,37] COPER [33]
LISP functions	LISP function	IO list pairs	restricted examples and COND function	rich	+	B	[65]
robot procedures	directed graph	execution traces	conditionals and non-counting loops	rich	+	I	NODDY [9,10]

Manuscript received ...

Bruce A. MacDonald, member
Ian H. Witten, member
Knowledge Sciences Institute, Department of Computer Science
The University of Calgary, 2500 University Drive NW, Calgary, Canada T2N 1N4

Acknowledgements

This research is supported by the National Sciences and Engineering Research Council of Canada. We gratefully acknowledge the stimulating research environment provided by the Calgary Machine Learning Group, and wish to thank in particular Rosanna Heise, Brent Krawchuk, Dave Maulsby, and David Pauli.

FOOTNOTES

1. Thimbleby [67] points out that some of the confusion arises because human perception of time—especially when you are thinking—is heavily distorted, so 31×52 *seems* the same thing as 1612, because you don't notice how long it really takes.
2. The interesting question of whether *either* are akin to the human variety of learning is not addressed in this paper.
3. This is the classic example used to contrast the PROLOG logic programming language with the LISP functional language.
4. But see [2,3,5,6,7,8,41,42] for some work in this area.
5. However, some research has been reported on learning initial knowledge from scratch [2,3,5,6,7,8,41,42].