88·3

Machine Learning, Meta-Reasoning and Logics

Workshop Proceedings

edited by

Pavel Brazdil

Sesimbra, Portugal

February 1988

INCREMENTAL LEARNING FROM EXAMPLES IN A LOGIC-BASED FORMALISM

Igor Mozetic & Nada Lavrac
Jozef Stefan Institute
Jamova 39, 61000 Ljubljana
Yugoslavia

ABSTRACT

In our view, a concise formalization of induction is essential for the advancement of research in machine learning. Formal logic provides a good basis for the knowledge representation and formalization of inference, both inductive and deductive. However, there are many subsets of first order predicate calculus (FOPC) that offer different tradeoffs between the expressive power and the efficiency of inference. In deductive systems, such as Prolog, a Horn clause subset of FOPC enables efficient interpretation. On the contrary, in inductive learning, systems of practical interest usually do not use any more expressive representation than an attribute-based language.

In the paper we propose to use the deductive hierarchical database formalism for knowledge representation. This formalism enhances the expressive power of an attribute-based language while allowing the implementation of an efficient learning algorithm and a more compact representation of complicated concepts. The algorithm for learning from examples and its incremental version are described in detail. The learning program is used as a module in a system for automatic construction of qualitative models. The system was successfully tested on a difficult problem of constructing a qualitative model of the heart.

1. INTRODUCTION

The task of inductive learning is to induce general descriptions of concepts from examples of these concepts. Examples are objects of a known class typically described in terms of attributes and values. The final product of learning are symbolic descriptions expressed in high-level, human understandable terms and forms. Induced descriptions of concepts, representing different classes of objects, can be used for classification, prediction and can account for new observations. Such a new representation of the domain knowledge may be very interesting to experts, especially in domains that are not well formalized and are poorly understood.

In the field of learning from examples several interesting practical methods and systems were developed. The majority of practically successful inductive learning systems use the attribute-based representation. These systems mostly belong to the families of the AQ (Michalski 1969) and ID3 (Quinlan 1983) learning algorithms. Recently, steming from ID3, a number of practical knowledge acquisition tools were developed in form of inductive expert system shells such as Assistant 86 (Cestnik et al. 1987), ExTran (Michie 1986), SuperExpert, etc. They can be directly used for knowledge acquisition of complex knowledge bases for expert systems, e.g. (Lavrac et al. 1986, Michie 1986).

From the point of view of formal logic the knowledge representation formalism of the above systems is equivalent to the propositional calculus (PC). They use the language of attributes and values to induce concepts descriptions in the form of if-then rules or decision trees. PC is the simplest logic representational formalism with the smallest expressive power. There is also a number of interesting inductive methods based on a Horn clause logic such as MIS (Shapiro 1981) and Marvin (Sammut & Banerji 1986) that induce simple formulas and concepts within this particular subset of first order predicate calculus (FOPC). Unfortunately it turns out that the computational complexity of these systems allows inducing only simple concepts. The reason for that is the search space of descriptions that is too large to allow induction of complicated concepts. Such systems, using a more expressive language, are unfortunately not yet practical for knowledge acquisition in expert systems.

In our view, formal logic provides a good basis for the knowledge representation and formalization of inference, both deductive and inductive. Logic has clear, declarative semantics and (deductive) inference can be characterized as theorem proving. Even if one does not want to use logic directly for knowledge representation (e.g., prefers semantic nets or frames), it is important that the selected representational formalism can be characterized in terms of FOPC. For example, an observation about the semantic nets form is that it contains only unary and binary predicates.

As noted by Levesque (1984) there exists a fundamental tradeoff in expressive power and tractability of inference in different knowledge representations. This tradeoff underlies the differences among a number of formalisms and motivates many research issues in knowledge representation. The chosen formalism has to be interesting both from the point of what it can represent and from the point of the reasoning strategies it permits.

It is worth noting that the role of logic for knowledge representation is different than in mathematics. In mathematics, the main role of FOPC is in the formalization of infinite collections of entities. On the other hand, in knowledge representation, the domains being characterized are usually finite. Thus the main role of a suitable subset of FOPC is to compactly express concepts and to deal with incomplete knowledge. Its expressive power specifies not so much what can be said, but rather how compactly can knowledge be expressed and what can be left unsaid (Levesque 1984).

In standard logic, the deductive machinery does not have any knowledge about the intended interpretation of nonlogical symbols. A remedy is to divide the individuals in the intended interpretation into different sorts and then specify the sorts of all nonlogical symbols in the language; this is known as many sorted or typed logic. Typed logic thus provides a simple syntactic way of specifying semantic information and can increase deductive efficiency by eliminating useless branches of the search space (Cohn 1987). However, the typing mechanism often reduces the expressive power of the language (as is the case with strongly typed programming languages).

By an 'increased expressive power' of a formalism we do not mean that there are concepts which cannot be expressed, as is the case when we say that FOPC is more expressive than PC. Rather, we mean that it is easier to express some concepts, that they can be directly represented, and in a more compact way. This is the case when we say that full FOPC is more expressive than clausal form, and the later is more expressive than Horn clauses. We should emphasize that there appears to be a tradeoff between expressiveness and efficiency in mechanised deductive systems (Cohn 1987). As the language becomes more restricted (FOPC, clausal form, Horn clauses) so the deductive inference becomes more efficient. Prolog (Clocksin & Mellish 1981) owes much of its success to having found a usefull tradeoff in this scale.

We argue that the same expressiveness/efficiency tradeoff that exists in deductive systems appears to exist in inductive learning systems as well. The aim of our work is to improve expressiveness whilst maintaining efficiency in inductive inference. In the paper we propose the use of the deductive hierarchical database formalism (Lloyd & Topor 1985) for knowledge represenation in a system for learning from examples. The formalism is based on a typed language thus resembling an attribute-based language, but enhancing its expressiveness by allowing the use of universally quantified variables, functors and utility predicates in rules. However, the formalism does not allow for recursively defined predicates and functors, which

substantially reduces the search space in induction. From the pragmatical point of view this is not a significant restriction since in machine learning problem domains are usually finite.

In section 2 the deductive hierarchical database formalism and corresponding inference rules are precisely defined. Its relation to other logic-based formalisms is characterized in terms of expressiveness. Section 3 formalizes the notions of generality and inductive inference. We state assumptions on which our inductive learning algorithm is based. The learning algorithm that uses the proposed knowledge representation formalism and its incremental version are described in section 4. The algorithm is based on the AQ inductive learning algorithm (Michalski 1969, Michalski et al. 1986). It is intended to be used in domains that are not noisy although they may be incomplete with regard to the set of attributes used to describe a domain. The system was successfully tested on a complex medical problem of learning the model of the electrical activity of the heart (Mozetic 1987a, b).

2. LOGIC-BASED KNOWLEDGE REPRESENTATION

In this section some basic logic programming (Lloyd 1984) and deductive database concepts (Lloyd & Topor 1984, 1985, 1986) are presented. Properties of the deductive hierarchical database formalism (DHDB) are discussed. It is argued that the DHDB formalism provides a framework in which expressive power of the existing inductive learning systems may be enhanced without decreasing the efficiency of learning.

2.1. Deductive hierarchical database formalism

A database clause is a typed first order formula of the form

$$A \leftarrow L_0, \ldots, L_N \qquad (1)$$

where A is an atom and each L_i is a literal (an atom or the negation of an atom). A is called the head and the conjunction of literals $L_0,\ldots,L_N$ the body of the clause. The body may be empty in which case the clause is called a fact. All variables are implicitly assumed to be universally quantified in front of the clause. A fact with no variables is called a ground fact. A deductive database is a finite set of database clauses. It is called hierarchical if its predicates can be partitioned into levels so that the bodies of the clauses in the definitions of higher level predicates contain only lower level predicates (Lloyd & Topor 1985).

The set of all database clauses with the same predicate in the head forms a predicate definition. As a predicate definition

consists of clauses that have the form of implication (1) the definition of the predicate specifies only the sufficiency condition for A to hold. Such a form allows the system only to infer positive information (i.e., A is true). In order to be able to deduce negative information as well (i.e., not A is true) the so called database 'completion' is to be performed. By this process a predicate definition is transformed into an equivalence. Although in a deductive hierarchical database system the form of implications is retained it is usually permitted and useful to regard the database as being implicitly completed (Lloyd 1984).

The deductive database formalism uses a typed language. It turns out that types provide a natural way of specifying the domain of a database. We emphasize that, in contrast to the usual restriction in deductive databases, in the proposed formalism functors are allowed to appear in a database and queries. The major reason for not allowing functors is that they can cause the set of answers to a query to be infinite. However, under a reasonable restriction of allowing only hierarchical types (the same restriction as for a hierarchical database) we can ensure that each query can have at most finitely many answers. This restriction bans recursive data types, which means that there are only finitely many ground terms of each type, and consequently, that each query can have at most finitely many answers (Lloyd & Topor 1985). This also means that the DHDB formalism is equivalent to propositional calculus in the sense of what can be expressed in the two formalisms. However, having variables, functors and rules, the DHDB form enables more compact representation of concepts.

The use of a database system based on first order logic for knowledge representation has the attractive property that it has a well-developed theory and offers a uniform formalism to represent facts, rules and queries. Further, answering a query may be regarded as a logical deduction and can be efficiently implemented using a Prolog-like system as the query evaluator. The only inference rule used is SLDNF-resolution which is a linear resolution augmented with the negation as failure rule. To ensure that the negation is handled properly (which is not the case with standard Prolog systems) a safe computation rule must be used that only selects negative literals that are ground. In this case it can be shown that the inference rule is sound, i.e., that all answers to a query are logical consequences of a completed database (Lloyd & Topor 1985). For a hierarchical database the inference rule is also complete, i.e., all logical consequences can be deduced from the completed database (Lloyd & Topor 1986). Since logical consequence and deduction are equivalent in the case of DHDB, in the sequel we use the symbol |= to denote both.

2.2. Expressive power of logic-based formalisms

When deciding about what logic formalism to choose for knowledge representation, one has to consider thoroughly the expressiveness/efficiency tradeoff, i.e., the tradeoff between what can be represented and how efficient are the reasoning strategies. Below is a list and a brief overview of the logic-based formalisms that are potential candidates to be used in a machine learning system.

(1) propositional calculus
(2) attribute-based language
(3) relational database formalism
(4) deductive hierarchical database formalism
(5) deductive database formalism
(6) Horn clause form
(7) clausal form
(8) first order predicate calculus

On one hand, we have the propositional calculus (PC) which has the smallest expressive power, and on the other, first order predicate calculus (FOPC) represents the strongest first order language. Other formalisms are sorted in the increasing expressiveness between the two extremes. By 'expressiveness' we again mean 'how' compactly and comprehensively can knowledge be expressed and not 'what' can be expressed in the chosen formalism. With regard to 'what' can be expressed, the formalisms (1) to (4) above are equivalent to PC, but with regard to 'how' they are substantially different. The following is a brief overview of the formalisms.

(1) Propositions in PC are 0-ary predicates. There are no variables and terms.

(2) In an attribute-based language attributes may be considered as unary predicates and their values as their arguments. Predicates are typed and domain closure and unique names axioms are assumed to apply (Gallaire, Minker & Nicolas 1984). No variables and functors are used.

(3) In the relational database formalism clauses are restricted to ground atoms. N-ary relations are represented by n-ary predicates; variables, functors and rules are not allowed. This formalism is equivalent to an attribute-based form, where arguments of a relation correspond to individual attributes.

(4) A deductive hierarchical database introduces variables, functors and non-recursive rules. Thus it is substantially more general than a relational database since it does not contain only ground facts but also non-ground facts and rules from which

new facts can be deduced. The language is typed, but only non-recursive types are allowed. Hierarchical databases is less general than a deductive database.

(5) A deductive database allows also recursive predicate definitions and recursive types (functors). With regard to the use of negation, database clauses are more general than Horn clauses since they allow for negation to be used in the body of a clause (implemented by the negation as failure rule).

(6) A Horn clause is a clause with at most one positive literal. Negative literals in the body are not allowed. The language is not typed.

(7) The clausal form allows disjunction of positive literals. Existential quantification is not allowed.

(8) FOPC allows also existential quantification of variables.

In the next section we distinguish between the attribute-based and the 'relational' notation. Note that (2) and (3) above are equivalent since individual attributes correspond to arguments of a relation. Relational notation may also be used in (4) which is an extension of (3).

2.3. Relation between attribute-based and relational notation

To illustrate the advantages (and some disadvantages) of the relational notation and the DHDB formalism over an attribute-based language, let us consider an example. The following are three rules that determine (in qualitative terms) the weight of an object from (inprecise and incomplete) information about its dimensions and shape (a two-dimensional view):

```
Weight = small  <=                                          (a.1)
   Shape = circle v square, Length = small                  (a.1.1)
         v
   Shape = rectangle, Length = small,                       (a.1.2)
   Width = small v medium

Weight = medium  <=                                         (a.2)
   Width = medium                                           (a.2.1)
         v
   Length = small, Width = large                            (a.2.2)
         v
   Length = large, Width = small                            (a.2.3)

Weight = large  <=                                          (a.3)
   Length = large, Width = large                            (a.3.1)
```

Rules are written in the VL_1 like language (Michalski 1974), which is used by most of the AQ-based learning algorithms. VL_1 is a representative of an attribute-based language. In VL_1, a selector relates an attribute to a disjunction of values (called internal disjunction), e.g.:

Shape = circle v square

A set of selectors forms a complex, e.g.:

Shape = circle v square, Length = small

A disjunction of complexes forms a cover. A description induced by an AQ based algorithm has the form of an if-then rule where the condition has the form of a cover, and the conclusion is a decision class.

In an attribute-based language such as VL_1, the domain closure and unique names axioms are usually assumed to apply. Now, a question of the precise meaning of those rules arises. If a set of selectors is interpreted as a logical conjunction, and the symbol '<=' as a logical implication, difficulties are easily encountered. If we have an object with the following properties:

Shape = rectangle, Length = small, Width = medium

the conditions of both rules (a.1) and (a.2) are satisfied, from which it follows:

Weight = small, Weight = medium |= small = medium

This is a contradiction since it violates the unique names axiom.

We propose to interpret if-then rules as relations and to write them in the DHDB formalism in the relational notation. The following is a precise interpretation of if-then rules we propose:

(1) '<=' is a predicate symbol written in the infix notation.
(2) An if-then rule for each individual complex corresponds to a DHDB clause.
(3) Attributes are arguments of the predicate.
(4) Attribute values correspond to the argument values.
(5) The internal disjunction is expressed by a binary utility predicate ':='.

In our example, we can replace '<=' by a 4-ary predicate 'weigth' (in the standard prefix notation). The following are rules (a.1, a.2, a.3) written in the relational notation:

```
% weight( Shape, Length, Width, Weight )
  weight( Shape, small,  _,     small )  <-          (r.1.1)
     Shape := circle v square.
  weight( rectangle, small, Width, small ) <-        (r.1.2)
     Width := small v medium.

  weight( _, _,     medium, medium ).                (r.2.1)
  weight( _, small, large,  medium ).                (r.2.2)
  weight( _, large, small,  medium ).                (r.2.3)

  weight( _, large, large,  large ).                 (r.3.1)
```

The underline character above denotes an anonymous variable. Now, let us assume that there is a definition of a utility predicate for (qualitative) multiplication:

```
  times( X,      X,      X ).
  times( X,      Y,      medium )  <-  ~( X = Y ).
  times( X,      medium, X ).
  times( medium, X,      X ).
```

Then instead of having six clauses (r.1.1, ..., r.3.1) in our example, only two suffice:

```
  weight( Shape, Length, _, Length )  <-
     Shape := circle v square.
  weight( rectangle, Length, Width, Weight )  <-
     times( Length, Width, Weight ).
```

An attribute-based and a relational notation are equivalent if we restrict ourselves to finite domains. Each notation has its own advantages and disadvantages, and their relation is similar to the relation between an n-ary and a binary representation (Kowalski 1979). We can take the VL_1 language as a representative of the attribute-based notation, and DHDB as a representative of the relational notation. The following are some differences between both notations:

(1) In VL_1 it is easier to add a new n+1st attribute. In DHDB a n-ary predicate has to be replaced by an (n+1)-ary predicate.
(2) In VL_1 attributes are explicitely named, and their names are global. In DHDB only variable names may denote attributes, and are local to each clause.
(3) In VL_1 an attribute with an unknown value may be ommited, while in DHDB an anonimous variable must be used.
(4) The advantage of DHDB over VL_1 is its well-defined semantics and existance of an efficient inference rule.
(5) In DHDB there are universally quantified variables. Expressing relations between them (using utility predicates) allows more compact representation than in VL_1.
(6) The use of functors is more natural in DHDB than in VL_1.

3. FORMALIZATION OF INDUCTIVE INFERENCE

A first order theory consists of an alphabet, a language, a set of axioms and inference rules (Lloyd 1984). The alphabet and syntax of the language define all well-formed formulas of the theory. The axioms are a designated subset of well-formed formulas. Using inference rules one can derive theorems of the theory from the axioms. For example, in section 2 we defined the DHDB formalism as a language and SLDNF-resolution as the corresponding inference rule.

Semantics (meaning) of a language is defined by an interpretation of its symbols. More precisely, an interpretation of the language L consists of the following:

(1) A set D, the domain of the interpretation.
(2) An assignement of each constant in L to an element in D.
(3) An assignement of each n-ary function in L to a mapping from n-tuples in D to single elements in D.
(4) An assignement of each n-ary predicate in L (denoting a relation in D) to a mapping from n-tuples in D into truth values + or -.

Each formula denotes a statement in the domain, while the interpretation defines its meaning and truth value (+ or -). For a given set of formulas (e.g. axioms), we are particularly interested in interpretations for which the formulas express true statements in the domain, and are assigned a truth value +. Such an interpretation is called a model. Normally, when we formulate a set of axioms, we already have in mind a distinguished interpretation which gives the principal meaning of the symbols. Such an interpretation is called the intended interpretation, and naturally, it should be a model.

Informally, the inductive task is to find a theory that will account for all specific observations in the domain under consideration. The theory should be as simple as possible in order to be easy to understand. Further, the theory should be general so that it can be used for prediction and can account for new observations.

3.1 Definition of generality

We say that a $Theory_1$ is more general than a $Theory_0$ iff $Theory_0$ is a logical consequence of Theory1 and they are not equivalent. We write:

$$Theory_1 \models Theory_0$$

Further, $Theory_1$ should be internally consistent, i.e. it has to have at least one model. This definition means that $Theory_1$ has more consequences and less models than $Theory_0$. Our definition

of generality corresponds to the intuitive notion in the case when there is a fixed domain in which one wants to identify as much properties and relationships between objects of the domain as possible. A theory from which more properties of the domain can be deduced is considered to be more general. In this context, the inference from more specific to more abstract is called the induction.

On the contrary, one may regard the notion of a theory being more general when it can be applied to more domains. In this case a more general theory has more models and is a logical consequence of any more specific theory. In this context, either the deductive inference may be used for generalization or one has to reformulate the theory to capture all potentially interesting domains.

3.2. Definition of inductive task

The inductive task is to find a theory with the intended interpretation not yet completely known. The intended interpretation (that is also a model) assigns a truth value + to all formulas denoting true statements in the domain. In the sequel we identify a model only by a set of all ground atoms that are assigned a truth value +. Each interpretation is therefore identified by a model and a pre-interpretation (Lloyd 1984, p.71). A pre-interpretation of a language consists of a domain, and assignements of elements and mappings in the domain to constants and functions in the language, respectively. Corresponding to a fixed pre-interpretation, there will in general be numerous interpretations based on it obtained by further specifying the model.

We assume that the intended interpretation is already partially specified by the user and that the inductive task is to complete what is not yet known. Specifically, we assume that a typed language and its pre-interpretation on which the intended interpretation is based, are already given. Note that the use of a typed language reduces possible interpretations by reducing the space of possible ground atoms in the language. We are not aware of any inductive learning algorithm using first order logic (or a Horn clause subset), e.g. (Shapiro 1981, Sammut & Banerji 1986, Buntine 1986b) that is using a typed language.

The search for the intended interpretation is further restricted by given facts, i.e., learning examples that assign truth values to some ground atoms. Possible models are only those that contain all ground atoms assigned a truth value + (positive learning examples), and do not contain any ground atom assigned a truth value - (negative examples).

In induction, we are actually searching only for axioms of the theory since inference rules are usually known. Further, some axioms of the theory may already be provided in the form of background knowledge, and are assumed true in the intended interpretation. This may simplify the theory, but it increases the search space of possible sets of axioms.

We formulate the inductive task as follows. Given are inference rules, a typed language and its pre-interpretation, a set of examples, and some background knowledge in the form of utility axioms. The task is to find a theory (Theory) from which, together with utility axioms (Util), all positive examples (PosExam) logically follow, and no negative example (NegExam) is its logical consequence. Choose the simplest between such theories. An initial, incomplete and incorrect theory may be already provided to guide the search.

Given: typed language and pre-interpretation of the intended interpretation I,
PosExam in the model M of I,
NegExam not in the model M of I,
Util background knowledge, true in I

Find: Theory such that:

Theory, Util $\models$ PosExam
Theory, Util $\not\models$ NegExam

In the sequel we restrict ourselves to the DHDB language and SLDNF-resolution as a single inference rule. In this formalism, axioms are represented by predicate definitions. Background knowledge is represented by utility predicate definitions, and axioms we are looking for are called inductive hypotheses.

4. INDUCTIVE LEARNING ALGORITHM

The inductive task is to find an inductive hypothesis that is complete and consistent with the given examples and takes into consideration the available background knowledge (Michalski 1983). In the DHDB formalism an inductive hypothesis has the form of a predicate definition (a set of clauses with the same predicate in the head). A learning example is a pair <A, ±>, where A is a ground atom and ± denotes its membership in the model M of the intended interpretation. We write:

<A, +> if A $\in$ M, a positive example
<A, -> if A $\notin$ M, a negative example

We say that a hypothesis is complete iff it covers all positive examples. A hypothesis is consistent with learning examples iff

it does not cover any negative example. A hypothesis covers an example iff any of its clauses covers the example. We say that a clause covers an example <A, ±> iff A is a logical consequence of the clause and the relevant utility predicates (Buntine 1986a). Formally, if there is a set of utility predicates (Util), then a clause:

$$A' \leftarrow L_0, \ldots, L_N$$

covers an example <A, ±> iff there is a substitution θ that unifies A with A', such that:

$$(L_0, \ldots, L_N)\theta, \text{Util}$$

is satisfiable.

4.1. The algorithm

The inductive learning algorithm proposed in this paper is based on the AQ algorithm (Michalski 1969). The AQ algorithm learns concept descriptions in the form of if-then rules, where the conclusion of a rule is a decision class, and the condition is an expression, called a cover. A cover consists of a disjunction of complexes, where each complex is a set of selectors (attributes with values). The algorithm first selects a positive example of a class and generalizes it into a complex by assigning all possible values to all attributes. The complex is then specialized so that it does not cover any negative example. The algorithm keeps several alternative complexes which maximize the number of covered positive examples and are as simple as possible. When no negative example is covered the algorithm selects the best complex and adds it to the current cover. The algorithm repeats the procedure until all positive examples are covered. The only specialization operation used in the algorithm is removing values from the internal disjunction of individual attributes.

The algorithm that finds an inductive hypothesis in the DHDB formalism is as follows:

```
Initial hypothesis is empty;
while the hypothesis is incomplete
      (does not cover all positive examples)
do    1. select an uncovered positive example (a seed);
      2. find a set of alternative covering clauses that
         cover the seed and no negative example;
      3. select the best covering clause and add it
         to the inductive hypothesis;
trim the inductive hypothesis.
```

The algorithm that finds a set of alternative covering clauses for a seed (step 2 above) is as follows:

Initial set consists of one covering clause
that is a generalization of the seed;
while any clause in the set is inconsistent
(does cover some negative example)
do 1. select a covered negative example;
2. specialize all inconsistent clauses to uncover the negative example, but still cover the seed;
3. remove those worst covering clauses that exceed the number of permitted alternatives.

Generalization of the seed is a maximally general clause. This is a unit clause where all arguments in the head are distinct variables.

Specialization of a clause with respect to the seed (that has to be covered) and the negative example (that should not be covered) can be done in one of the following ways:

1. replace a variable in the head by a compound term (with respect to the variable type),
2. replace a variable in the head by an internal disjunction of constants (with respect to the variable type),
3. remove some constants from an internal disjunction,
4. unify two distinct variables in the head (if they are of the same type),
5. add a utility predicate or its negation to the body of the clause (with respect to types of arguments); this is called constructive induction.

At the end, a complete and consistent inductive hypothesis is trimmed in two steps:

1. in each clause the number of constants in internal disjunctions is minimized, subject to the constraint that all positive examples are still covered;
2. all clauses that do not cover uniquely at least one positive example are removed from the hypothesis.

A measure of quality for a clause determines a partial order of alternative clauses and is a parameter defined by the user (in the AQ algorithm it is called LEF). This constitutes, apart to the language syntax, an essential part of the inductive bias used in the system. The measure consists of a list of criteria that are applied to each clause. When the first criterion cannot discriminate between two clauses the second one is used, and so on. The following is a list of default criteria that order clauses from the best to the worst:

1. higher number of covered positive examples,
2. lower number of literals in the body of the clause,

3. lower total number of constants in internal disjunctions and non-variable terms in the head of the clause.

The second parameter, definable by the user is the maximum number of alternative clauses kept by the algorithm at any time (called maxstar in AQ). Default value for this parameter is 10.

4.2. Incremental learning

The algorithm guarantees that an inductive hypothesis is complete and consistent with all learning examples considered so far, but it may not be true in the intended interpretation. If new learning examples are encountered which indicate that the hypothesis is not true, there are two possibilities. The hypothesis can be discarded and a new hypothesis can be learned from all, original and new learning examples. On the other hand, the hypothesis may be incrementally refined.

The incremental algorithm is similar to the basic algorithm, with the exception that it does not start with an empty hypothesis. Initially, all inconsistent clauses (that cover some negative examples) are removed from the initial hypothesis. Next, the algorithm finds clauses that cover all positive examples not yet covered or no longer covered. And finally, those covering clauses are added to the modified initial hypothesis.

For the incremental version of our algorithm it is important that the algorithm does not discard any learning examples after the inductive hypothesis is found since they might be needed for further refinement. This is called full memory incremental learning (Reinke & Michalski, in press). Another approach to incremental learning is to discard all original learning examples and to retain a weighted initial hypothesis, from which learning examples are generated (Lee & Ray 1986). Obviously, this approach does not guarantee the completeness and consistency of inductive hypothesis with the original learning examples.

The second characteristic of our incremental algorithm is that it does not specialize inconsistent clauses (but discards them), nor does it generalize consistent clauses. This is quite the opposite to the GEM incremental learning algorithm (Reinke 1984, Reinke & Michalski, in press), which is another implementation of the AQ algorithm and a predecessor to our algorithm. However, our algorithm is more efficient than GEM for that reason, and incrementally learned hypotheses are simpler.

In our view, the essential advantage of incremental learning over non-incremental is in its greater efficiency. The time complexity of the AQ algorithm is proportional to the number of all negative examples and clauses (thus indirectly to the number

of positive examples) in the final inductive hypothesis (Skorstad 1986). If the algorithm is to generalize initial consistent clauses it has to check their consistency with respect to all, original and new learning examples. It follows from this that the time complexity would be the same as for non-incremental learning. The reason that our algorithm does not specialize inconsistent clauses (which would increase its efficiency), but rather discards them from the initial hypothesis is to avoid their over-specialization.

At the moment, the algorithm is implemented partially in Prolog and partially in Pascal. The part in Prolog (cca. 1000 lines) transforms learning examples from DHDB formalism to an attribute-based language VL_1, and induced rules from VL_1 back to DHDB formalism. The core of the algorithm is the NEWGEM program, implemented in Pascal (cca. 5000 lines). NEWGEM is another implementation of the AQ algorithm (Mozetic 1985) that learns rules from examples and uses the VL_1 language. For the purposes of our algorithm, any algorithm that learns the inductive hypothesis in the attribute-based language (e.g., generates a decision tree), like the ID3 algorithm (Quinlan 1983) could be used as well.

5. CONCLUSION

The majority of practically applicable inductive learning systems use the attribute-based language for knowledge representation. On the other hand, there are some interesting inductive methods based on a Horn clause subset of FOPC that are not yet of practical interest for knowledge acquisition in expert systems. The paper proposes the DHDB formalism for knowledge representation and describes the incremental learning algorithm that works in this formalism. We argue that the DHDB formalism provides a framework that improves expressiveness whilst maintaining efficiency in inductive inference. Feasibility of the learning system was tested on a complex medical problem. It was used to infer a qualitative model of the electrical activity of the heart from sets of examples of the heart components behaviour (Mozetic 1987a, b).

Further work in the development of the inductive learning system is planned along several directions. In order to improve the efficiency of induction an ID3-based inductive algorithm will be incorporated into the DHDB formalism. We plan to extend the typed language to handle polymorphic types and arbitrary type lattices (Cohn 1987), thus providing features that are very powerful for dealing with taxonomic inference and are known in semantic nets and frames. Finally, we plan to enhance the constructive induction feature, by inducing stuctured concept descriptions with meaningful intermediate concepts.

ACKNOWLEDGEMENTS

A part of the research described here was completed during the first author's stay at the AI Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, USA. This research was supported in part by the National Science Foundation under Grant No. DCR 84-06801, the Office of Naval Research under Grant No. N00014-82-K-0186, the Defense Advanced Research Project Agency under Grant No. N00014-K-85-0878, the Slovene Research Council, and by the COST-13 project.

REFERENCES

Buntine, W. (1986a). Generalized subsumption and its applications to induction and redundancy. Proc. European Conference on Artificial Intelligence, Brighton, UK, July 21-25.

Buntine, W. (1986b). Induction of Horn clauses: methods and the plausible generalization algorithm. Technical report, School of Comp. Sc., NSW Institute of Technology, Broadway, Australia.

Cestnik, B., Kononenko, I., Bratko, I. (1987). ASSISTANT 86: a knowledge elicitation tool for sophisticated users. In Progress in Machine Learning (I.Bratko, N.Lavrac, Eds.), Sigma Press, Wilmslow, UK.

Clocksin, W.F., Mellish, C.S. (1981). Programming in Prolog. Springer-Verlag.

Cohn, A.G. (1987). A more expressive formulation of many sorted logic. Journal of Automated Reasoning 3 (1).

Gallaire, H., Minker, J., Nicolas, J.M. (1984). Logic and databases: a deductive approach. Computing Surveys 16 (2).

Kowalski, R. (1979). Logic for Problem Solving. North Holland.

Lavrac, N., Varsek, A., Gams, M., Kononenko, I., Bratko, I. (1986). Automatic construction of a knowledge base for a steel classification expert system. Proc. 6th Intl. Workshop on Expert Systems and their Applications, Avignon, France.

Levesque, H.J. (1984). A fundamental tradeoff in knowledge representation and reasoning. AI report 35, Schlumberger, Palo Alto, CA.

Lee, W.D., Ray, S.R. (1986). Rule refinement using the probabilistic rule generator. Proc. 5th National Conference on Artificial Intelligence, Philadelphia, PA, August 11-15, Morgan Kaufmann.

Lloyd, J.W. (1984). Foundations of Logic Programming. Springer-Verlag.

Lloyd, J.W., Topor, R.W. (1984). Making Prolog more expressive. Journal of Logic Programming 1 (3).

Lloyd, J.W., Topor, R.W. (1985). A basis for deductive database systems. Journal of Logic Programming 2 (2).

Lloyd, J.W., Topor, R.W. (1986). A basis for deductive database systems II. Journal of Logic Programming 3 (1).

Michalski, R.S. (1969). On the quasi-minimal solution of the general covering problem. Proc. 5th Intl. Symposium on Information Processing (FCIP 69), Vol. A3, Bled, Yugoslavia.

Michalski, R.S. (1974). VL1: variable-valued logic system. Intl. Symposium on Multiple-Valued Logic, West Virginia University, Morgantown, WV.

Michalski, R.S. (1983). Theory and methodology of machine learning. In Machine Learning: An Artificial Intelligence Approach (R.S.Michalski, J.G.Carbonell, T.M. Mitchell, Eds.), Tioga, Palo Alto, CA.

Michalski, R.S., Mozetic, I., Hong, J., Lavrac, N. (1986). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. Proc. National Conference on Artificial Intelligence, Philadelphia, PA, August 11-15, Morgan Kaufmann.

Michie, D. (1986). Machine learning and knowledge acquisition. In Automating Knowledge Acquisition (D.Michie, I.Bratko, Eds.), Addison Wesley.

Mozetic, I. (1985). NEWGEM: Program for learning from examples, technical documentation and user's guide. Technical report ISG, Dept. of Comp. Sc., University of Illinois at Urbana-Champaign, Urbana, IL.

Mozetic, I. (1987a). Learning of qualitative models. In Progress in Machine Learning (I.Bratko, N.Lavrac, Eds.), Sigma Press, Wilmslow, UK.

Mozetic, I. (1987b). The role of abstractions in learning qualitative models. Proc. Fourth International Workshop on Machine Learning, Irvine, CA, August 22-25, Morgan Kaufmann.

Quinlan, J.R. (1983). Learning efficient classification procedures and their application to chess end games. In Machine Learning: An Artificial Intelligence Approach I (R.S.Michalski, J.G.Carbonell, T.M.Mitchell, Eds.), Tioga, Palo Alto, CA.

Reinke, R. (1984). Knowledge acquisition and refinement tools for the ADVISE meta-expert system. M.Sc. thesis, Dept. of Comp. Sc., University of Illinois at Urbana-Champaign, Urbana, IL.

Reinke, R., Michalski, R.S. (in press). Incremental learning of concept descriptions. In Machine Intelligence 11 (J.E.Hayes, D.Michie, J.Richards, Eds.), Oxford University Press.

Sammut, C., Banerji, R.B. (1986). Learning concepts by asking questions. In Machine Learning: An Artificial Intelligence Approach II (R.S.Michalski, J.G.Carbonell, T.M.Mitchell, Eds.), Morgan Kaufmann, Los Altos, CA.

Shapiro, E.Y. (1981). Inductive inference of theories from facts. Research Report 192, Yale University, New Haven, CT.

Skorstad, G. (1986). AQ complexity. Technical report ISG, Dept. of Comp. Sc., University of Illinois at Urbana-Champaign, Urbana, IL.