Machine Learning, , 209–240 (1997)
© 1997 Kluwer Academic Publishers. Manufactured in The Netherlands.

Integrating Multiple Learning Strategies in First Order Logics

A. GIORDANA attilio@di.unito.it
F. NERI neri@di.unito.it
L. SAITTA saitta@di.unito.it
M. BOTTA botta@di.unito.it
Dipartimento di Informatica, Università di Torino, C.so Svizzera 185, 10149 Torino, Italy

Editors: Ryszard S. Michalski and Janusz Wnek

Abstract. This paper describes a representation framework that offers a unifying platform for alternative systems, which learn concepts in First Order Logics. The main aspects of this framework are discussed. First of all, the separation between the hypothesis logical language (a version of the VL_{21} language) and the representation of data by means of a relational database is motivated. Then, the functional layer between data and hypotheses, which makes the data accessible by the logical level through a set of abstract properties is described. A novelty, in the hypothesis representation language, is the introduction of the construct of internal disjunction; such a construct, first used by the AQ and Induce systems, is here made operational via a set of algorithms, capable to learn it, for both the discrete and the continuous-valued attributes case. These algorithms are embedded in learning systems (SMART+, REGAL, SNAP, WHY, RTL) using different paradigms (symbolic, genetic or connectionist), thus realizing an effective integration among them; in fact, categorical and numerical attributes can be handled in a uniform way. In order to exemplify the effectiveness of the representation framework and of the multistrategy integration, the results obtained by the above systems in some application domains are summarized.

Keywords: Learning Relations, Multistrategy Learning, Learning from Databases

1. Introduction

Learning knowledge expressed in First Order Logic (FOL) has been an appealing task since the beginning of Machine Learning. Early attempts (Plotkin, 1970, Winston, 1975, Hayes-Roth & McDermott, 1978, Vere, 1978, Michalski, 1980, Dietterich & Michalski, 1983, Kodratoff & Ganascia, 1986) have proposed a number of conceptually interesting ideas, which provided foundations and suggestions for later work. However, the need of excessive computational resources made the proposed methods rather impractical. This drawback even arose the question whether it was worthwhile at all to devote efforts to the development of systems for learning in FOL; in fact, when dealing with a finite, function- and recursion-free universe (which was the most frequent case for classification problems), FOL and propositional calculus are equivalent. On the other hand, this very equivalence allows knowledge to be represented in FOL in a much more compact and readable form, which lets relations emerge, which could be difficult to discover when dispersed in a possibly large number of ground propositions.

Moreover, on the application front, real domains are usually affected by noise, and objects are often described by continuous attributes, all of this adding to the basic complexity of handling FOL formulas *per se*. Notwithstanding the above difficulties, some systems, such as ML-SMART (Bergadano et al., 1988, Bergadano et al., 1991) and FOCL (Pazzani & Kibler, 1992) have shown that FOL concept learning is not only abstractly interesting, but feasible. The system ML-SMART, in particular, has been applied to a number of real world problems (Bergadano et al., 1991, Giordana et al., 1993b), and suggested effective solutions for many of the encountered problems. More precisely, the system had special mechanisms to handle noise and continuous attributes, it learned a structured knowledge base (not just a set of flat ”condition-action” rules), and it allowed for evidential reasoning. An important aspect of ML-SMART was its interface to a database, from which the examples were extracted, and in which both the generated hypotheses and their extensions were stored (Bergadano et al., 1988). This feature proved to be essential in learning an industrial troubleshooter (Giordana et al., 1993b) used in field.

ML-SMART evolved later into two new systems, both including inductive and deductive components: SMART+ (Botta & Giordana, 1993), which emphasizes handling noise and continuous attributes by exploiting powerful heuristics for controlling the search in the hypothesis space, and WHY (Baroglio et al., 1994, Saitta et al., 1993), which stresses the importance of exploiting background knowledge, in the form of a causal model of the application domain, and of learning comprehensible and justifiable knowledge. Moreover, both systems have an embedded machinery, based on relational algebra, for interacting with a relational database; this feature makes the systems particularly well suited to perform data mining and knowledge discovery in databases. More recently, a number of new learning applications in FOL have been reported (see, among others, (Bratko & Džeroski, 1995)).

The development of other systems (see, for instance, MOBAL (Morik, 1991) and RIGEL (Gemello et al., 1991)), and the theoretical work on generalization and complexity (Buntine, 1988, Haussler, 1988, Helft, 1989, Michalski, 1991, Flach, 1995) contributed to a deeper understanding of this difficult and challenging task.

In the last years, learning in FOL has been re-proposed as ”Learning Relations” (Quinlan, 1990) and Inductive Logic Programming (ILP) (Muggleton, 1991). Originally, learning relations, as exemplified by the system FOIL (Quinlan, 1990), was an extension to FOL of the top-down construction of a decision tree, similar to the method presented in (Bergadano & Giordana, 1988). Novelties, with respect to previous FOL learning approaches, were, on one hand, the possibility for the learned concept to contain variables, i.e., to be the name of a relation, an intentional definition of which was to be found from its extensional representation, and, on the other hand, the possibility of learning recursive concepts.

ILP stems from two sources: its goal recalls Shapiro's work, aimed at synthesizing logic programs (Shapiro, 1983), whereas its theoretical background focuses on logical theories of induction. Actually, as acknowledged in (Ade et al., 1995), most ILP systems have dealt with concept learning from examples, so that the differences between ILP and previous research in learning in FOL loose sharpness. In addition, some non-ILP learning systems can easily be extended in order to learn recursive concepts, as the system RTL (Giordana et al., 1993a, Baroglio & Botta, 1995); on the other hand, such an extension may not be so easy for other systems, as shown in (Cameron-Jones & Quinlan, 1993).

Whatever the name under which learning in FOL is performed, a major issue to be tamed is computational complexity. Learning existentially quantified concepts has been proved to be an NP-complete problem (Haussler, 1988) even in very simple cases. There are basically two ways to deal with this problem: the first is to reduce the search by adding various kinds of bias in the learning process (Gordon & desJardins, 1995), both declarative (syntactic and semantic (Ade et al., 1995)) and procedural ones. The second one is to increase the search power of the learning algorithm (Muggleton, 1995). Constraining the search by adding a strong declarative bias, aimed at reducing the expressive power of the hypothesis language, is the solution most frequently adopted in ILP (Ade et al., 1995). Systems like SMART+, instead, have a much weaker declarative bias, but they widely exploit procedural biases in the form of search heuristics (Botta & Giordana, 1993).

Increasing the search power has been tried for the first time (in FOL) in the system REGAL, which learns relations via a genetic algorithm (Giordana & Sale, 1992, Giordana & Saitta, 1994, Giordana & Neri, 1996, Neri, 1997). Genetic search for learning concepts from examples has been used, previously, only in the context of propositional logic (Grefenstette et al., 1990, McCallum & Spackman, 1990, Bala et al., 1991, De Jong et al., 1993, Greene & Smith, 1993, Janikow, 1993, Vafaie & De Jong, 1991) . By its very nature, learning in FOL has traditionally been cast inside the symbolic paradigm. However, the possibility of learning or revising relational knowledge by means of other paradigms, such as the genetic or the connectionist one, could greatly enlarge the class of solvable problems.

The aim of this paper is neither to re-describe the learning systems we developed, nor to present an extensive experimentation: the interested reader may find detailed descriptions in previous papers. Rather, the goal is to illustrate some of the key issues emerged in developing different systems inside alternative paradigms, and in trying to solve some real-world problems (Botta et al., 1992, Giordana et al., 1993b, Faure et al., 1993, Saitta et al., 1995, Baroglio et al., 1996, Neri & Saitta, 1996). In particular, we will concentrate on the multi-strategy design aspects involving:

- integration between symbolic and numeric information
- integration among different paradigms (symbolic, genetic and connectionist)
- cooperation between tools and concepts from machine learning and database methodologies.

Other equally relevant issues, such as effective exploitation of a domain theory (Bergadano & Giordana, 1988, Botta & Giordana, 1993), or cooperation among different reasoning mechanisms (Saitta et al., 1993, Baroglio et al., 1994) will be left aside for the sake of sharpening the paper's focus.

The most fundamental issue for an effective achievement of the three mentioned types of integration is knowledge representation. In particular, both discrete and continuous-valued descriptors have to be handled in a homogeneous way (at least from the user's point of view). Moreover, any type of involved knowledge must be dealt with by systems based on different paradigms without any need of translation. Finally, in view of the application on large databases, a learning system must be able to transparently, quickly and effectively interact with (commercial) database management systems.

In this paper, we will describe a knowledge representation framework that shows the above features. We would like to stress that this framework is only a subset of the one used by the systems we developed: each of them, when working stand-alone, can make use of wider representation languages, well-suited for its own specific approach. For instance, SMART+ extends the hypothesis language with numerical quantification, and exploits a domain theory expressed as a set of Horn clauses. The presented framework is, in some sense, the intersection of the knowledge representation facilities of the various systems, specifically tailored for a multistrategy approach. In fact, a number of systems, differing with respect to the paradigm they are based on, and for the task they are oriented to, can share this same framework, leaving the user free of selecting the learning system according to its suitability to the problem and not to the type of representation it can handle.

In particular, the framework has several strong points. The first one is the ability to deal with formulas containing the logical construct of internal disjunction. The importance of such a construct, in order to obtain compact and readable knowledge, has been underlined by Michalski since his first works (Michalski, 1980, Michalski & Chilausky, 1980, Michalski, 1983), in which he introduced the VL_1 and VL_{21} languages. In this paper we will present efficient algorithms for learning internal disjunctions.

Another aspect to be stressed is the decoupling between the logical level of hypothesis representation and the relational database format in which the examples are stored. The link between the two levels is supplied by a mapping, containing a set of semantic functions, which acts as an interface between the database and the learner. In this way, the learner gives to the interface a formula to verify and receives back the answer, ignoring how the data are represented. This solution has the advantage that performing data mining on a database does not require rewriting the examples in an ad-hoc format (e.g., ground clauses). On the other hand, such a separation between hypotheses and data allows different data representations to be used, by simply changing the semantic mapping.

Finally, the above two features, namely the availability of the internal disjunction and the transparency of data representation achieved through the semantic mapping, allow both discrete and continuous attributes to be dealt with in a uniform way by the learning algorithms.

2. The Knowledge Representation Framework

Many learning algorithms in FOL describe the learning instances using a subset of the hypothesis description language in ground form. This was the method adopted, for instance, by Induce (Michalski, 1980, Michalski, 1983), and, more recently, by GOLEM (Muggleton & Feng, 1990). This method has the advantage that the truth of a formula in the hypothesis description language can be proved using logical resolution only. In the framework we present, on the contrary, we keep separate the hypothesis description language (HDL) and the instance description language (IDL). The HDL consists of a declarative definition of the logical language syntax used for expressing hypotheses. The IDL, instead, includes the capability of representing in a declarative way both the internal *structure* of the examples and their *properties*.

In other words, the IDL allows an instance to be represented not only as a compound object, but also in terms of a collection of properties that may be meaningfully used (with respect to the application at hand) "to reason about" the instances. Then, a complete representation of an instance consists of the specification of its elementary component objects, plus a set of *properties* such as "color of an object", "area of an object", or "relative position (follows/precedes) between two objects", which, together, describe the *structure* of the instance; this representation is close to the style adopted in object oriented databases. The ability to handle property definitions in the IDL is one of the key notions to understand where the peculiar advantages of the proposed representation come from. Also, this is a major point where IDL differs from just a structural representation of the instances by means of a list of ground predicates.

Hypotheses are expressed in a logical language, whose basic predicate semantics is defined in terms of the instance properties specified in the IDL. This representation framework allows different reasoning schemes to be used at the hypothesis and at the instance levels, simplifying the integration of different learning algorithms; moreover, its modularity makes it easy to interface a learning system (working at the hypothesis level) to a database, without any preprocessing of the dataset.

2.1. The Instance Description Language

When learning propositional knowledge, it is common practice to describe the learning instances as vectors of attributes. We will now extend this framework in order to deal with structured learning instances (or examples) and their properties.

Before describing the structural representation of the examples, we need to define the meaning of *concept instance* and of *learning instance* (or *scenario*) in our framework. Given a *concept*, i.e. a name of a relation, a *concept instance* is a group of objects, that, as a whole, can be labeled with that name. More formally, a concept is a n-ary predicate $c(x_1, \ldots, x_n)$, and a concept instance is an n-tuple $< a_1, \ldots, a_n >$ of objects such that $c(a_1, \ldots, a_n)$ is true. A *scenario*, which is a novelty with respect to previous representation schemes, is simply a collection of atomic objects. As an analogy, we may think of a scenario as a segmented picture. A scenario may contain several instances of the same concept or even of different concepts. The main characteristic of a scenario is that it constitutes a separate universe, in the sense that the variables occurring in a formula can only be bound to objects occurring in the same scenario. For the sake of illustrating the above notions, let us consider Figure 1, where six scenarios from a block world domain are depicted. Let L be a language containing the following set of literals:

$$P = \{Height(x, j), Length(x, j), Color(x, k), Follows(x, y)\}, \tag{1}$$

where x and y must be bound to an element of the set $\{1, 2, 3, 4, 5, 6\}$, j to an element of $\{1, 2, 3\}$, and k to an element of $\{b, w, d\}$. Thus, predicate $Height(x, j)$ is true if the height of block x is j, predicate $Follows(x, y)$ is true if x follows y in the sequence, and so on. Let us now consider a concept $\omega(x, y)$ defined as follows:

$$\begin{aligned} &Height(x,3) \wedge Color(x,b) \wedge Height(y,1) \wedge \\ &\quad \wedge Color(y,d) \wedge Follows(y,x) \rightarrow \omega(x,y) \end{aligned} \tag{2}$$

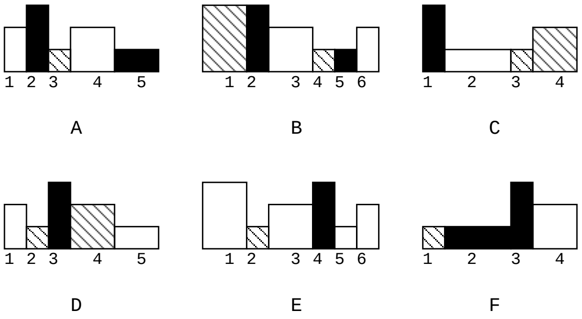

Figure 1. A toy block world domain. Scenarios A, B, C contain positive instances of the allomorphic concept $\omega(x,y)$, defined by (2), whereas scenarios D, E, F do not contain any.

Description (2) can be phrased as follows:
"In ω there is a high, black block, followed by a short dashed block".

Three positive instances of concept $\omega(x,y)$, namely the pairs of blocks (2,3) in A, (2,4) in B and (1,3) in C, exist. On the other hand, no instances of ω exist in scenarios D, E and F. If we now consider the concept $Black(x)$, such that:

$$Color(x,b) \rightarrow Black(x) \tag{3}$$

then, all the scenarios in Figure 1 contain some instances of it; in particular, scenario A contains two instances, blocks 2 and 5. Moreover, block 2 is (part of) an instance of both $\omega(x,y)$ and $Black(x)$.

The logical architecture of the dataset has then three levels of abstraction: scenarios, concept instances, and elementary objects. Each elementary object (simply called "object" in the following) has a vector of attributes associated to it. The restriction bound variables only inside a scenario strongly limits the combinatorial explosion of the size of relations (extensions) associated to hypotheses. According to this view, the best would be to generate one scenario for each concept instance. But, in this case, relations among different instances, such as those involved in recursive definitions, could not be found. Another possibility is to adopt the opposite solution, i.e., to consider a unique scenario, containing all the instances, and to limit the search by imposing syntactic restrictions on the occurrence of variables (this is one of the reasons for the introduction of *modes* associated to the variables of a predicate in some ILP systems). A scenario can be considered as a kind of semantic declarative bias (Ade et al., 1995).

Let us now consider the concrete representation of a scenario. Let us keep in mind that a scenario is a set of atomic objects possibly belonging to different concept instances. In Figure 2, all of the objects of Figure 1 are represented in a relation with 6 fields, corresponding to the four attributes (Length, Height, Color and Position), plus two additional

Objects

S	ObjId	Length	Height	Color	Pos
A	1	1	2	w	1
A	2	1	3	b	2
A	3	1	1	d	3
A	4	2	2	w	4
A	5	2	1	b	5
B	1	2	3	d	1
B	2	1	3	b	2
B	3	2	2	w	3
B	4	1	1	d	4
B	5	1	1	b	5
B	6	1	2	w	6
C	1	1	3	b	1
C	2	3	1	w	2
C	3	1	1	d	3
C	4	2	2	d	4
D	1	1	2	w	1
D	2	1	1	d	2
D	3	1	3	b	3
D	4	2	2	d	4
D	5	2	1	w	5
E	1	2	3	w	1
E	2	1	1	d	2
E	3	2	2	w	3
E	4	1	3	b	4
E	5	1	1	w	5
E	6	1	2	w	6
F	1	1	1	d	1
F	2	3	1	b	2
F	3	1	3	b	3
F	4	2	2	w	4

$Black^*(x)$

S	x
A	2
B	2
B	5
C	1
D	3
E	4
F	2
F	3

$\omega^*(x, y)$

S	x	y
A	2	3
B	2	4
C	1	3

Figure 2. Relation describing the atomic objects of the dataset in Figure 1.

attributes: S identifies the scenario an object belongs to, and ObjId identifies an object inside a scenario. These two attributes are used as primary and secondary index keys to fastly address the objects in the relation. In the relation $Objects$, objects are listed without reference to the concept(s) they are instances of: $Objects$ only describes the segmentation of the scenarios. The information of what objects belong to which concept instance is kept in separate relations, each one corresponding to a concept. As an example, let us consider the concept ω, described by expression (2): this concept has arity 2 and a relation $\omega^*(x, y)$ can be associated to it, as in Figure 2. The first row in relation ω^* states that objects 2 and 3 of scenario A constitute a positive instance of ω. If we consider concept $Black(x)$, the corresponding relation (also reported in Figure 2) contains all the black blocks occurring in the various scenarios. Negative examples are implicitly defined using the Closed World Assumption (Clark, 1978): every tuple not occurring in a relation describing a target concept is a negative example of it. Then, in our framework, we adopt the same non-monotonic semantics as De Raedt and Džeroski (De Raedt & Džeroski, 1994).

The same scheme easily handles the problem of representing multiple target concepts. In this case it is sufficient to define a relation for each one of them. Finally, heterogeneous objects, with different types, can be stored in different $Objects$ relations.

Let us now consider the definition of object properties. Available domain knowledge could be exploited for this task. As previously said, properties are defined using a functional declaration, as in object oriented programming.

Let x be a complete object description (i.e., a row of the $Objects$ relation). A valid property is essentially a (boolean, discrete, or continuous) function having as domain the cartesian product $x^n (n \geq 1)$. Let us denote with *BOOL, DISC, CONT* a boolean, discrete, or continuous property, respectively. From the implementation point of view, each property definition is a computational expression on a set of pre-defined functions providing access to the object attributes. Examples of property definitions for the objects in Figure 2, are the following [1]:

$$\begin{array}{ll} high(x) :: x.Height > 2 & (BOOL) \\ color(x) :: x.Color & (DISC) \\ height(x) :: x.Height & (CONT) \\ area(x) :: x.Height * x.Length & (CONT) \\ distance(x_0, x_1) :: |x_1.Pos - x_0.Pos| & (CONT) \end{array} \tag{4}$$

where *x.Height* denotes the value of the attribute Height of the object bound to variable x.

It may happen that a property $p(x_1, \ldots, x_n)$ cannot be expressed in analytical form in terms of existing object attributes; in this case, the property is directly represented in extensional form in the database by means of a corresponding relation $p^*(x_1, \ldots, x_n, v)$. To handle this case, we introduce a property whose value v is directly extracted from relation p^*. For instance, the property "percentage of overlapping between x and y" may be expressed as:

$$overlap\_perc(x, y) :: \begin{cases} v & if(x, y) \in overlap^*(x, y, v) \\ 0 & otherwise \end{cases} \tag{5}$$

2.2. The Hypothesis Description Language

As for IDL, we will first describe the Hypothesis Description Language from an abstract point of view, and, then, we will show, step by step, how a formula of HDL can be evaluated, in an efficient way, on the adopted learning instance representation by applying a sequence of *relational algebra* operators (Ullman, 1982). The HDL is a clausal language in annotated predicate calculus, equivalent to the VL_{21} language (Michalski, 1983) without numerical quantifiers but with negation on atoms. Moreover, the HDL we used allows predicates to contain one internal disjunction. Even though a more complex version of this language, including numerical quantifiers and unrestricted negation has been used in SMART+ (Botta & Giordana, 1993) and Enigma (Giordana et al., 1993b), we will consider here this simpler form that allows a direct comparison with the languages used in other systems such as FOIL, FOCL and most ILP programs.

The reasons for choosing an annotated predicate calculus as HDL, instead of classical Horn clauses, are the compact and readable hypothesis representation (thanks to the internal disjunction construct), the uniform representation of predicates dealing with discrete or continuous attributes, and the easiness in evaluating the predicates on the learning instances (represented in IDL). In the following, after introducing the annotated predicate calculus, each of the above reasons is discussed in detail.

According to the definition given by Michalski (Michalski, 1980, Michalski, 1983), in annotated predicate calculus a term occurring in a predicate can be a constant, a variable, or a disjunction of constants. The atomic formula

$$Color(x, White \vee Black \vee Yellow) \tag{6}$$

is well formed in annotated predicate calculus. As it is intuitive, the meaning of (6) is "the color of object x is either white, or black, or yellow". It is also immediate to verify that formula (6) is equivalent to the disjunction:

$$Color(x, White) \vee Color(x, Black) \vee Color(x, Yellow). \tag{7}$$

In general, every clause in annotated predicate calculus can be syntactically rewritten into an equivalent set of classical clauses, preserving its semantics. Notice how some constructive induction mechanism (Wnek & Michalski, 1994), such as that of Fringe (Pagallo, 1989), builds new predicates on disjunctive atomic expressions, so reproducing something similar to what is immediately available in annotated predicate calculus.

Predicates in HDL are of two kinds: *constraint predicates*, which do not contain the internal disjunction term, and *learnable predicates,* containing an internal disjunction. The latter derive their name from the fact that their internal disjunction term can be modified during the learning process. Internal disjunctions are essentially sets of constants of the same type: symbolic (discrete) or real (continuous). For the sake of notation compactness, we denote finite sets of discrete or symbolic values by $[v_1, \ldots, v_n]$, and infinite sets, corresponding to intervals along unidimensional real space, by $\langle v_1, v_2 \rangle$.

The predicate semantics is defined in terms of object properties expressed as described in the previous section. Specifically, the semantics of *constraint predicates* is defined as

$$CP(x_1, \ldots, x_n) :: BOOL_{CP}(x_1, \ldots, x_n) \tag{8}$$

where $BOOL_{CP}$ denotes any boolean property of the objects. The semantics of *learnable predicates,* based on a discrete internal disjunction, is defined as

$$LP(x_1, \ldots, x_n, [v_1, \ldots, v_n]) :: member(DISC_{LP}(x_1, \ldots, x_n), [v_1, \ldots, v_n]) \tag{9}$$

In (9) $DISC_{LP}$ denotes a discrete property, and the function $member$ is a boolean function which returns "true" if the property value belongs to the set $[v_1, \ldots, v_n]$. In an analogous way, for continuous intervals, the predicate semantics is defined using a definition of the type:

$$LQ(x_1, \ldots, x_n, \langle v_1, v_2 \rangle) :: inside(CONT_{LQ}(x_1, \ldots, x_n), \langle v_1, v_2 \rangle) \tag{10}$$

where the function $inside$ checks whether the property value belongs or not to the specified interval. Notice that the specification of the semantics of a predicate on an extensionally defined property is covered by one of cases (8), (9) or (10).

As said above, the considered HDL allows single atom negation. However, the negation of an atomic formula of type (9) or (10) semantically means that the property value is not member of the set defining the internal disjunction. In other words, we can state the following equivalences:

$$\begin{aligned} &\neg LP(x_1, \ldots, x_n, [v_1, \ldots, v_n]) \leftrightarrow \neg member(DISC_{LP}(x_1, \ldots, x_n), [v_1, \ldots, v_n]) \\ &\neg LQ(x_1, \ldots, x_n, \langle v_1, v_2 \rangle) \leftrightarrow \neg inside(CONT_{LQ}(x_1, \ldots, x_n), \langle v_1, v_2 \rangle) \end{aligned} \tag{11}$$

Predicates may have any arity, but in many applications it turned out that binary ones are sufficiently expressive.

Finally, an *inductive hypothesis* is a formula of the type $\varphi(x_1, \ldots, x_m) \rightarrow R(y_1, \ldots, y_r)$ where $\varphi(x_1, \ldots, x_m)$ is a conjunction of predicates on variables $x_1, \ldots, x_m$, possibly containing internal disjunctions, and $R(y_1, \ldots, y_r)$ denotes a target concept of arity r, where $y_1, \ldots, y_r$ is a subset of $x_1, \ldots, x_m$. In the HDL, the range-restriction bias is assumed (De Raedt & Džeroski, 1994), i.e., variables occurring in the head of a clause must also occur in the body. By referring to the examples of Figure 1, some well formed hypotheses in HDL may be:

$$\begin{array}{l} Height(x_0, \langle 2.5, 4 \rangle) \rightarrow HighRect(x_0) \\ Color(x_0, [b]) \wedge Color(x_1, [d]) \wedge Follows(x_1, x_0) \rightarrow P(x_0, x_1) \end{array} \tag{12}$$

The complete description of a concept usually consists of a disjunction of a set of inductive hypotheses.

Let us now consider the operational mechanism supporting the evaluation of a hypothesis in HDL. As described in Section 2.1, the learning instances are stored in database relations. Then, considering the inductive hypothesis $\varphi(x_1, \ldots, x_m) \rightarrow R(y_1, \ldots, y_r)$, to check for its consistency (completeness) means to verify that the extension of $\varphi(x_1, \ldots, x_m)$, projected on the variables $y_1, \ldots, y_r$, is included into (includes) the extension of $R(y_1, \ldots, y_r)$. The extension of $\varphi(x_1, \ldots, x_m)$ can be computed by applying a sequence of selection and natural join operators, as usual in relational databases.

Referring to the examples in Figure 1, suppose we want to evaluate the extension on $Objects$ of the formula $Color(x_0, [b]) \wedge Color(x_1, [d]) \wedge Follows(x_1, x_0)$. From the relation $Objects$ the extensions ${\psi_1}^*$ of formula $\psi_1(x) = Color(x, [b])$ and ${\psi_2}^*$ of formula $\psi_2(x) = Color(x, [d])$ can be obtained by selecting (*selection operator*) from the relation $Objects$ all the objects having color $black$, i.e., satisfying the condition $member(x.Color, [b])$, and $dashed$, respectively. The two corresponding relations are the first two tables in Figure 3. Afterwards, the extension φ^* of formula $\varphi(x_0, x_1) = Color(x_0, [b]) \wedge Color(x_1, [d]) \wedge Follows(x_1, x_0)$ can be generated, starting from the two relations ${\psi_1}^*$ and ${\psi_2}^*$, by first applying a *natural join* operator, which generates the intermediate relation ${\psi_3}^*$, corresponding to formula $\psi_3(x_0, x_1) = Color(x_0, [b]) \wedge Color(x_1, [d])$, and then a *selection operator* on ${\psi_3}^*$ in order to eliminate the tuples that do not satisfy the condition $Follows$ (i.e., $x_1.Pos > x_0.Pos$). The process is illustrated in the second two tables in Figure 3. Every conjunctive formula in the considered HDL can be evaluated with a similar procedure[2].

3. A Framework for Integrating Multiple Learning Strategies

In this section we will show how the HDL language, defined in the previous section, lends itself to the integration of induction algorithms with complementary abilities, hence offering a powerful multistrategy framework. First of all, we will briefly recall how inductive inference, specified via a high level strategy, can be realized on a database by means of *relational algebra* (Ullman, 1982). The main advantage of this approach, already discussed in

$Color(x, [b])$

ψ_1^*

S	x
A	2
A	5
B	2
B	5
C	1
D	3
E	4
F	2
F	3

$Color(x, [d])$

ψ_2^*

S	x
A	3
B	1
B	4
C	3
C	4
D	2
D	4
E	2
F	1

$Color(x_0, [b]) \wedge Color(x_1, [d])$

ψ_3^*

S	x_0	x_1
A	2	3
A	5	3
B	2	1
B	2	4
B	5	1
B	5	4
C	1	3
C	1	4
D	3	2
D	3	4
E	4	2
F	2	1
F	3	1

$Color(x_0, [b]) \wedge Color(x_1, [d]) \wedge Follows(x_1, x_0)$

ψ_4^*

S	x_0	x_1
A	2	3
B	2	4
C	1	3
C	1	4

Figure 3. Extensional evaluation of a hypothesis φ by using a sequence of selection and natural join operations.

(Giordana et al., 1993b) and applied in various systems, is the possibility of directly embedding learning abilities into a database manager. This possibility, which is important for data mining applications, is a common platform for all the learning approaches exploiting the described HDL, and allows a natural cooperation among them. The considered approaches are the symbolic one, exploited in the systems SMART+, SNAP, WHY and RTL, the genetic one, in REGAL (Giordana & Saitta, 1994) and SMART+ (Botta & Giordana, 1993), and the connectionist one, in SNAP (Botta & Giordana, 1996).

The multistrategy cooperation framework is hierarchically structured. At the higher level, we pose a symbolic relational learner like SMART+, which is responsible for constructing a first order classification theory. The genetic and connectionist strategies work at the lower level by refining the internal disjunctions occurring in the hypotheses constructed by the symbolic learner.

In the following, we will describe how the above mentioned learning systems, even though requiring different internal representations, can effectively share the same HDL, and, in particular, how they can all deal with internal disjunction, whose advantages have been underlined in the previous sections. Moreover, we will describe methods for easily specifying, for each approach, declarative biases, both syntactic and semantic, when they are needed.

Finally, we will show how the introduced representation framework offers the means of handling in a uniform way both discrete and continuous attribute values occurring in the description of the examples.

3.1. The Relational Learning Component

The basic learning strategy considered in this paper is a top-down construction of a hypothesis tree: the nodes of the tree are hypotheses (i.e., formulas that represent intentional descriptions of relations), whereas the edges are labeled by inductive operators, based on the extended relational algebra described in Section 2. The growth of the tree along a branch stops when either a consistent hypothesis is found or when the formula associated to the leaf satisfies some heuristic halt criterion.

Every time a new formula is created and added to the tree, the corresponding extensional relation is created as well, and it is stored on a mass storage device. In this way the search algorithm keeps checkpoints of the states it went through, thus allowing more sophisticated search strategies, requiring a degree of backtracking, to be used. As a side-effect, the induction algorithm is robust, as it can easily run for long periods, if necessary, surviving possible operating system crashes. On the contrary, systems such as FOIL and many ILP systems use a simpler hill climbing strategy, because they can only keep one growing hypothesis at a time during search.

The search tree is generated step by step, and the following actions are repeated at each step:

1. Select from the frontier of the tree a set Φ of formulas φ that are good candidates for specialization by addition of new literals.

2. Determine the set of literals $\mathbf{P}$ which can be added to each $\varphi \in \Phi$ according to domain knowledge and heuristic criteria.

3. Generate a set Ψ of new formulas ψ by AND-ing each $\varphi \in \Phi$ with a literal $P \in \mathbf{P}$, selected according to some heuristic evaluation.

4. For every $\psi \in \Psi$ that is consistent on the learning set, declare covered the instances verifying ψ, update the frontier and restart.

The process stops when either all positive instances are covered or no more promising formulas are in the frontier. Detailed examples of hypothesis tree generations can be found in (Bergadano et al., 1991).

As it appears from the algorithm, many heuristics and biases are involved, which have been widely investigated in other papers. In principle, any heuristics suggested in the literature can be used within this framework without any special restriction. Moreover, since the software structure is very robust and capable of dealing with large datasets, also other search strategies, such as best-first or hill-climbing, can be implemented, instead of the beam search strategy discussed in the algorithm.

The use of relational algebra parallels, at the extensional level, the intentional search algorithm, and generates a tree, whose nodes contain the extensions of the relations associated to the corresponding intentional nodes. This approach requires large storage resources, whereas it may reduce the time to evaluate the generated hypotheses. On the contrary, by avoiding keeping the extensional relations, no large space resources are needed, but the time to evaluate new hypotheses, made on demand, may be much larger. On the other hand, with the current low-cost, large-capability storage devices, even large trees can be built up. For instance, with a tree of 1000 nodes, each associated to a relation with 5 variables and 20000 tuples, the amount of required memory is of the order of 200 Mbytes, which is entirely acceptable.

A kind of semantic bias similar to *determinacy* (Cameron-Jones & Quinlan, 1993, Ade et al., 1995) is used by ML-SMART and the other systems of its family, in order to avoid the generation of hypotheses having too large extensions. More precisely, let m be the number of instances covered by a hypothesis $\varphi \rightarrow \omega$. Let, m^* be the number of tuples (models) in relation φ^*. The hypothesis $\varphi \rightarrow \omega$ will be generated iff $m^*/m < \tau$, being $\tau \geq 1$ a user definable parameter. By setting $\tau = 1$ we require that a hypothesis be determinate, whereas by setting $\tau > 1$, a weaker condition is required. Usually, value from 5 to 10 is used for τ.

Another aspect, characterizing the present framework, that needs to be explained is the way in which internal disjunction is dealt with. In particular, we have to face the problem of evaluating a literal as a candidate to be added to a formula φ. This can be seen as a specific learning subtask. In the following subsection, implemented solutions to this subtask will be described. All of them use the notion of *predicate template*, which plays a role similar to that of *schemata* (Morik, 1991) or *relational cliché* (Silverstein & Pazzani, 1991). Given a predicate P, containing an internal disjunction, the *template* T_P associated to P is the maximum internal disjunction allowed for P: any internal disjunction occurring in an instance of P can only be a subset of T_P. As an example, the template for the predicate $Color$ shall enumerate all considered color values. If the attribute has a continuous range of values, as, for instance, the predicate $Height$, the *template* shall denote the maximum range allowed for the values (Giordana & Saitta, 1994).

3.2. Learning Internal Disjunctions Locally

In the general-to-specific learning strategy described in Section 3.1 it has been assumed that an algorithm for determining the best predicate to add to a formula φ is available. When a candidate predicate contains an internal disjunction as one of its arguments, the "best" disjunction among all the possible ones is to be determined. For instance, in the predicate $Color(x, [yellow, red, blue])$ one of the possibilities is $Color(x, [yellow, blue])$.

We will briefly overview the algorithms used to this purpose by SMART+ and by SNAP. *Algorithm 1* learns discrete disjunctions, whereas *Algorithm 2* learns continuous intervals. In both cases, the problem is to evaluate a formula $\psi = \varphi \wedge P(.., V)$, obtained by AND-ing φ with a literal containing an internal disjunction V. The evaluation $\mu(\psi)$ of ψ is the weighted sum of two terms (Botta & Giordana, 1993):

$$\mu(\psi) = \alpha\mu_1(\varphi, \psi) + (1 - \alpha)\mu_2(\psi), \tag{13}$$

where $\mu_1(\varphi, \psi)$ is the information gain (Quinlan, 1990) of ψ with respect to φ, and $\mu_2(\psi)$ an evaluation of the absolute completeness and consistency of formula ψ. In SMART+, $\mu_2(\psi) = v(\psi) * w(\psi)$, being $v(\psi)$ the proportion of positive instances covered by ψ, and $w(\psi)$ the fraction of correct classifications made by ψ. The parameter α is user definable.

On the basis of this evaluation criterion, the induction algorithm, for determining the best discrete disjunction in a literal P to be added to a formula φ, works as in the following:

Algorithm 1

```
1. Let $V$ be the internal disjunction to be computed, initially empty.
2. For every item $v_i$ belonging to the template $T_P$, create the relation ${\psi_i}^*$ containing the
   extension of formula $\psi_i = \varphi \wedge P(..., [v_i])$ and rank it according to (13).
3. Initialize a relation $\psi^*$ with the content of the relation ${\psi_k}^*$ having the highest rank, and
   initialize $V = [v_k]$.
4. Let $\Psi^*$ be the set of all the relations $\{{\psi_i}^* | i \neq k\}$
5. While $\Psi^*$ is not empty, repeat:
   (A) Select the relation ${\psi_j}^* \in \Psi^*$ having the highest rank.
   (B) If the evaluation $\mu(\psi \vee \psi_j) \geq \mu(\psi)$, merge ${\psi_j}^*$ into $\psi^*$ and add $v_j$ to the set $V$.
   (C) Remove ${\psi_j}^*$ from the set $\Psi^*$.
6. Return $\langle V,\ \mu(\psi)\rangle$.
```

Even though *Algorithm 1* is not optimal, it is relatively fast, because it is linear with the number of v_i's, and in practice it gives good results.

In order to deal with continuous intervals a different algorithm has been designed (Botta & Giordana, 1993):

Algorithm 2

Let $V = \langle v_m,\ v_M\rangle$ be the template of the interval to be determined; let moreover k be an assigned integer, and $\Delta = (v_M - v_m)/k$.

```
1. Define $k + 1$ equally spaced points $\{v_i | 0 \leq i \leq k\}$ on the interval $V$ with $v_0 = v_m$
   and $v_k = v_M$.
2. Generate the set of $k(k + 1)/2$ different subintervals $\{\langle v_i, v_j\rangle | 0 \leq i \leq k - 1, 1 \leq j \leq
   k, i < j\}$.
3. Evaluate, according to rule (13), every segment $\langle v_i, v_j\rangle$ by instantiating formula $\psi_{ij} =
   \varphi \wedge P(.., \langle v_i, v_j\rangle)$.
4. Return the interval $\langle v_A, v_B\rangle$ which obtained the best evaluation.
```

The interval $\langle v_A, v_B\rangle$ can be further refined by adjusting limit v_A (v_B) in an interval $\langle v_A - \Delta, v_A + \Delta\rangle$ ($\langle v_B - \Delta, v_B + \Delta\rangle$) using the same algorithm, as sketched in Figure 4.

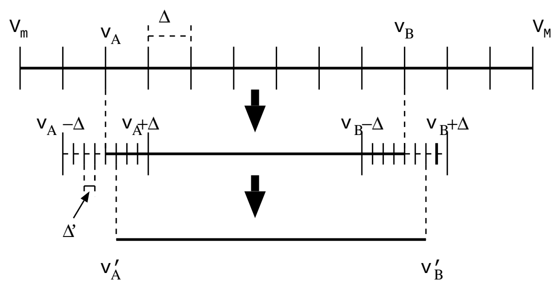

Figure 4. Iterative learning procedure for inducing an internal disjunction consisting of a continuous interval

3.3. Learning Internal Disjunctions Globally

The algorithms described in the previous subsection determine an internal disjunction by performing a local optimization, which accounts only for the literals already present in the current hypothesis. On the other hand, a global optimization of all internal disjunctions occurring in the predicates of a formula requires excessive computational resources, if dealt with classical optimization algorithms. However, the problem can be approached by using a Genetic Algorithm (GA) as a search engine. In the following we will describe how a formula with predicates containing an internal disjunction can be mapped to a bitstring, processable by a GA.

Let us suppose that some algorithm (or a human expert) has generated a hypothesis φ. In general, φ can be written as the conjunction $\varphi = \varphi_{CP} \wedge \varphi_{LP}$ of two formulas: φ_{CP}, containing constraint predicates (without internal disjunction), and φ_{LP}, containing learnable predicates (with internal disjunction).

Formula φ_{CP} states a set of necessary constraints, which need to be verified in order to correctly describe the target concept; on the contrary, formula φ_{LP} needs to be optimized with respect to the values occurring in its internal disjunctions. In order to achieve this goal, the following steps are performed:

1. From φ_{LP} a *Language Template* is generated, i.e. a formula having the same predicates as φ_{LP} but with the actual internal disjunctions replaced by the corresponding *templates*.

2. The language template defines the *chromosome* of individuals processed by the GA.

3. An initial population of individuals is generated from φ_{LP} by applying random mutations to its internal disjunctions.

4. The genetic algorithm is run until the best individual in the population remains stable for an assigned number of generations (or some other halt criterion is met).

The way internal disjunctions are encoded in a chromosome needs some explanation. In the case of a language template containing only discrete sets, it is immediate to map the elements of the different sets to a bit string. As shown in Figure 5(a), every item in a template corresponds to a boolean gene on the chromosome. When the gene assumes the value 1, the corresponding attribute value will be considered present in the corresponding internal disjunction, otherwise not. When an individual needs to be evaluated, the actual values for the internal disjunctions are loaded from the bit string into the template so creating a new formula which is matched on the learning instances.

This simple encoding method is currently used by the REGAL system, and a detailed description of it can be found in (Giordana & Saitta, 1994, Giordana & Neri, 1996).

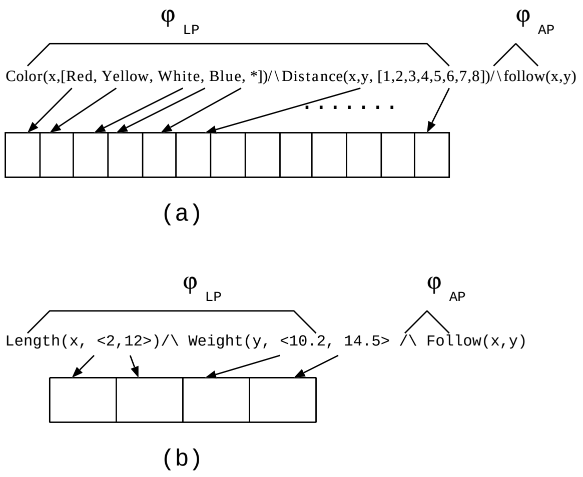
φ LP
φ AP
Color(x,[Red, Yellow, White, Blue, *])/\Distance(x,y, [1,2,3,4,5,6,7,8])/\follow(x,y)
(a)
φ LP
φ AP
Length(x, <2,12>)/\ Weight(y, <10.2, 14.5> /\ Follow(x,y)
(b)

Figure 5. Method for encoding in a bit string a *language template*: (a) internal disjunctions of discrete values; (b) continuous intervals. The star denotes the possible completion of a set of values.

For a continuous interval, defined by a pair of reals, a different approach has been adopted. A first solution, in agreement with the most classical GA approach (Goldberg, 1989), is to transform the real numbers into discrete integers that can be encoded as binary numbers; in this way, the chromosome becomes a bit string. This kind of transformation is described in (Goldberg, 1989) and has been used in a version of SMART+ (Botta & Giordana, 1993). A different approach can be taken, following the tendency of the Evolutionary Computation approach, using a chromosome composed of “real” genes (see Figure 5(b)). In this way, special crossover and mutation operators, capable of combining real numbers, must be used. For instance, in the (μ, λ)-model (Bäck, 1995), the crossover works as in the following:

1. Randomly select a pair of chromosomes.

2. Randomly select a gene position, say n.

3. A new chromosome is created by taking the genes in position different from n in part from one parent and in part from the other. Then, a new gene in position n is generated as average of the corresponding values in the two parents.

The mutation operator perturbates the value of a randomly selected gene according to a Gaussian distribution. This method is actually used in the most recent version of SMART+.

To deal with templates containing both discrete sets and continuous intervals is more complex. The solution of approximating real numbers with binary numbers can be easily integrated with the solution proposed for dealing with discrete sets, because the final chromosome structure is still a bit string. As an alternative, it is not difficult to conceive a GA capable of dealing with a hybrid chromosome, but this solution has not yet been experimented. Global optimization is used in two ways. Discrete optimization of internal disjunctions is actually the learning strategy of REGAL: learning is nothing else that a search for the "best" internal disjunctions in a language template. Continuous optimization of intervals is performed by SMART+ as a postprocessing of the formulas learned by the symbolic learner and containing numerical parameters. A few comments are in order regarding the template. The fixed length of the template is not a limitation on the type of hypotheses that can be learned. In fact, the formula φ, encoded in the template, may be generated by a symbolic learner, such as SMART+, which uses the full language defined in Section 2. As a special case, the template can be supplied directly by the user, when he/she has sufficient knowledge to establish the maximum reasonable complexity of the sought descriptions.

3.4. Dealing with Uncertainty and Refining a Theory by Performing the Quadratic Error Gradient Descent

A frequent problem in real world applications is the presence of uncertainty in the data. Boolean features, as they are learned by symbolic systems, have little reliability, because of the intrinsic variability of the phenomena they try to capture. In this sense, the neural network approach is considered much more effective, even if the networks look like black boxes, difficult to understand to the end user.

An attempt to combine the properties of neural networks with the ones of symbolic systems is represented by continuous valued logics, such as Fuzzy Logic and Probabilistic Logic, which have been successful in some applications. Nevertheless, the real power of neural networks, not yet available for continuous valued logics, derives from effective learning algorithms, such as back-propagation (Rumelhart & McClelland, 1986, Rumelhart et al., 1985). For this reason several methods have been proposed, trying to extend learning algorithms in such a way that the quadratic error gradient descent can be applied, as it has been done for propositional logic with neural networks (Towell et al., 1990, Opitz & Shavlik, 1993, Opitz & Shavlik, 1995, Berenji, 1992, Baroglio et al., 1996, Botta & Giordana, 1996).

Here we will follow a similar approach, but starting from the First Order Logic language we have used so far, instead of propositional logic. The aim is to exploit the abstraction properties offered by this framework in order to learn better and more compact descriptions for a "fuzzy concept". In (Baroglio et al., 1996, Blanzieri & Katenkamp, 1996, Tresp et al., 1993) it has been widely discussed how a propositional logic theory can be translated into a Radial Basis Function Network (RBFN) (Poggio & Girosi, 1990) that can be tuned following the quadratic error gradient descent. The underlying idea is very simple. A set of n continuous attributes defines an n dimensional space $\mathbf{X_n}$, where the ith attribute corresponds to an axis x_i in $\mathbf{X_n}$. In such a space, a classification theory can always be stated as a set of rules in the VL_1 language having, the general structure:

$$\varphi_j = x_1 \in A_{j1} \wedge x_2 \in A_{j2} \wedge \ldots \wedge x_n \in A_{jn} \rightarrow \omega \tag{14}$$

In (14) every A_{ji} is a segment parallel to the corresponding axis x_i, so that the intervals set $\{A_{ji} | 1 \leq i \leq n\}$ define a hypercube $\mathbf{A_j}$ inside the space $\mathbf{X_n}$. Rule (14) states the condition of membership in $\mathbf{A_j}$ for an attribute vector $\vec{x} \equiv \{x_1, ..., x_n\}$, in order to assert class (or concept) ω.

It is easy to give a continuous valued "fuzzy" semantics to this kind of rule, by replacing the boolean membership function in $\mathbf{A_j}$ with a continuous valued membership function, as it is done, for instance, in fuzzy logic (Zadeh, 1965, Zadeh, 1992). A standard solution is to replace every segment A_{ji} with a Gaussian function with width equal to A_{ji}. Interpreting the logical conjunction as an arithmetic product, the hypercube $\mathbf{A_j}$ is transformed into a Hypergaussian having the general format:

$$G_j(\vec{x}) = \prod_{i=1}^{n} g_{ji}(x_i) = \prod_{i=1}^{n} e^{-(\frac{x_i - c_{ji}}{A_{ji}})^2} = e^{-\sum_{i=1}^{n} (\frac{x_i - c_{ji}}{A_{ji}})^2} \tag{15}$$

According to (Poggio & Girosi, 1990), expression (15) is a Factorizable Radial Basis Function. Then, interpreting the logical (inclusive) disjunction as an arithmetic sum (Blanzieri & Katenkamp, 1996), a continuous valued semantics for a classification theory composed of rules like (14) can be defined through a network of radial basis functions of the type:

$$f(\vec{x}) = \sum_j w_j G_j(\vec{x}) = \sum_j w_j \prod_{i=1}^{n} g_{ji}(x_i), \tag{16}$$

where the weights w_j play the role of certainty factors. Expression (16) has a straightforward interpretation in terms of a four layer network, as described in Figure 6. The first layer (from left to right) represents the input features. The second layer units compute the unidimensional Gaussian functions g_{ji}. The third layer corresponds to rules, and combines the activation functions. Finally, the last layer computes the output value according to expression (16).

In the case of a multiclass classification theory, it is immediate to extend this neural architecture in order to have multiple output units, i.e., one for every class, as it is done in other neural networks (Rumelhart & McClelland, 1986).

Even though transforming a propositional theory into a RBFN does not preserve the original logical rigor, we obtain the important advantage that a RBFN can be finely tuned by means of the Δ-rule (Rumelhart & McClelland, 1986). In fact, expression (16) represents a continuous function, derivable with respect to the weights w_j, the centers c_{ji} and the widths A_{ji}, so that the quadratic error gradient can be computed everywhere, allowing the Δ-rule to be applied (Baroglio et al., 1996, Blanzieri & Katenkamp, 1996).

Here we will extend this method to the case of First Order Logics, so that a classification theory learned by a relational learner can be refined by performing the quadratic error gradient descent. In principle, also in this case the continuous intervals can be replaced

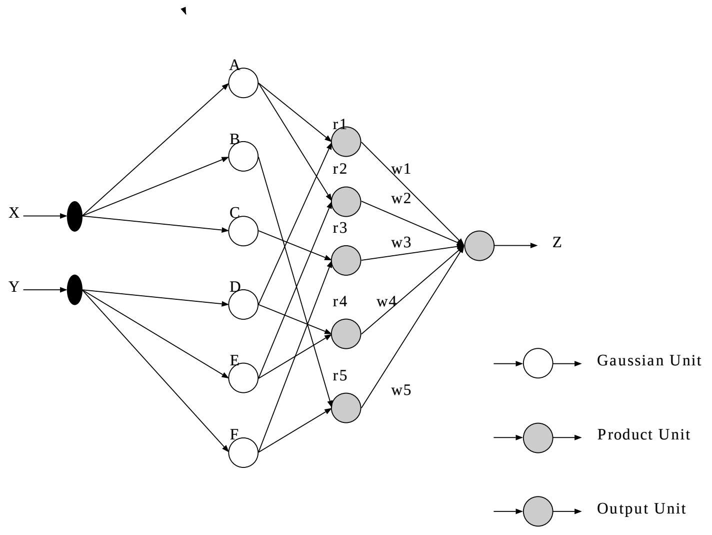

Figure 6. Network corresponding to a Factorizable RBFN architecture. The Gaussian units have a unidimensional activation function. The Product units compose the input values using arithmetic product. An average sum unit performs the weighted sum of the values received from the Product units, normalized with respect to the global activation of the network.

with bell-shaped functions, the logical conjunction with arithmetic product and the inclusive OR with arithmetic sum, obtaining thus a continuous valued semantics which is derivable in order to perform the error gradient descent. The substantial difference with respect to the propositional calculus is in the dynamic binding between variables occurring in the logical formulas and components of an instance.

A simple way of reformulating the classification problem in FOL, in such a way that it can be handled in the framework of a RBFN, is to consider every alternative binding as a different pattern, which feeds the network and undergoes classification independently. In this way, every single binding can be processed as in the case of the propositional calculus. In order to reach a correct classification, we must require that for each positive instance there exists at least one binding that activates the network above the classification threshold ($Th = 0.5$), and that no such binding exists for any negative instance. The classifier architecture obtained in this way is represented in Figure 7.

The drawback of this method lies in the possibly great number of alternative bindings. A solution we developed for predicting temporal series consists in restricting the bindings to only those corresponding to correct models[3] of the classification theory in the positive examples in the learning set. In this way, if the learning set contains at least one example for each one of the possible allomorphisms of the phenomenon described by the learning theory, we are guaranteed to generate all and only the bindings useful for the classification task. In several applications we performed, the number of useful different bindings turned out to be few tens.

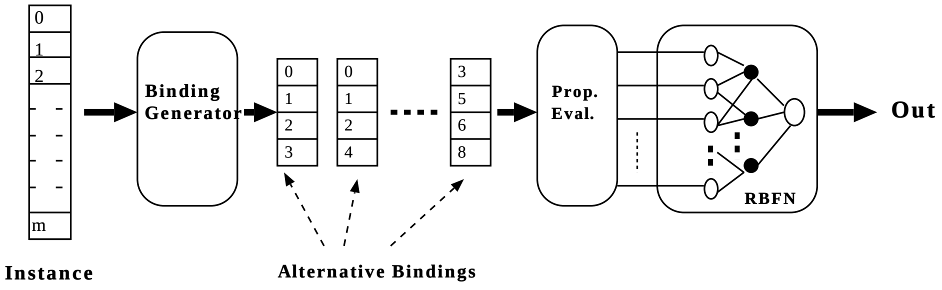

Figure 7. Three stage architecture for FOL predictor based on a RBFN. The first stage generates all the possible bindings; the second stage computes the object properties for each selected binding; the third stage accomplishes the classification.

A different mapping between FOL formulas and propositional calculus is presented in the system LINUS (Lavrač et al., 1991), where the atomic predicates are transformed into sets of positive and negative literals, which then play the role of attributes.

4. Summary of Experiments

In the previous section we have shown how different learning paradigms, i.e., the symbolic, the genetic and the connectionist ones, can be integrated in the same framework, exploiting the characteristics of a First Order representation language based on internal disjunction.

The aim of this section is neither to compare the results obtained by the learning systems we developed with those of other systems, nor to re-describe the extensive experimentation that is reported in previous papers. The goal is, instead, that of discussing the relative merits of the various paradigms and the easiness with which they can be integrated in a unique learning framework. The reported application examples are only a sample of those that have actually been performed. The results summarized in this section have been obtained with three systems, each one exploiting one or more of the algorithms described in Section 3:

- REGAL (Giordana & Sale, 1992, Giordana & Saitta, 1994, Giordana & Neri, 1996), which is based on a genetic algorithm and makes use of predicates containing only discrete internal disjunctions.
- SMART+ (Botta & Giordana, 1993), a multistrategy system *per se*, which combines symbolic learning strategies (both inductive and deductive) with the algorithms for learning internal disjunctions also in continuous domains. Moreover, SMART+ is provided with a postprocessor, based on a genetic algorithm, which accomplishes the refinement of the continuous intervals previously learned.
- SNAP (Botta & Giordana, 1996), a variant of SMART+, which has been designed for learning to predict temporal series, and combines the symbolic learning paradigm with the connectionist one, according to the method described in Section 3.4.

4.1. Office Document Classification

This first application shows the same problem solved independently by REGAL and SMART+. Even though the performances are similar in terms of recognition rate, the structure of the knowledge base generated by the two systems is very different. The problem consists in recognizing a company from the typical pattern appearing in a document. The dataset contains 230 examples, subdivided at the source into a learning set of 120 and a test set of 110 instances, corresponding to the front page of official documents from six different companies. Each example in the dataset is described as a sequence of 9 to 15 items, each corresponding to a specific rectangular area on the page layout. Moreover, each item is characterized by a type attribute, and by 8 numeric attributes individuating the coordinates of the four corners of the corresponding rectangle. Both SMART+ and REGAL have been supplied with a concept description language containing 5 unary predicates, one identifying the type and the other ones the position and the size of an item, and two binary predicates defining the relative position of two items. Each (absolute or relative) predicate was provided with an internal disjunction corresponding to the meaningful variability range of the corresponding numeric feature. However, REGAL does not have the capability of dealing with continuous intervals and copes with this problem by transforming a continuous interval into a set of discrete ones. In the specific case, the intervals have been split with a granularity of 35 pixels. Moreover, SMART+ did not receive as input any knowledge other than the set of predicates, whereas REGAL was supplied with a handcrafted language template to start with; this template is a very complex one, much more than that sufficient to represent the formulas usually found by SMART+.

REGAL discovered a classification theory consisting of 7 rules, correctly classifying all the learning examples. On the test set, the same rules exhibited a 5% misclassification error (1 omission error and 5 commission errors). SMART+ discovered a classification theory consisting of 52 rules. On the test set, the rules exhibited a 4% misclassification error (1 commission error less than REGAL). An attempt to further refine the internal disjunctions in the rules, using SMART+'s genetic postprocessor, did not lead, in this case, to any improvement. An evaluation of INDUBI (Esposito et al., 1992), an Induce-like learner, on a subset of this dataset shows an 8.5% prediction error. However, a direct comparison is not possible because these results have been obtained using a smaller learning set.

From the error rate point of view, REGAL and SMART+ look substantially equivalent; nevertheless, their generalization capabilities appear quite different. Even if they started from the same language, the knowledge base generated by SMART+ is much more detailed and complex than the one generated by REGAL. In fact, as discussed in (Giordana & Neri, 1996), REGAL has a large generalization capacity, which usually allows it to find classification theories very much compact and simple. On the contrary, SMART+ has other advantages, such as the possibility of guiding induction with large bodies of a priori knowledge. Nevertheless, SMART+'s local optimization strategy cannot compete, with respect to the ability of finding simple knowledge bases, with REGAL's global strategy. REGAL's generalization capacity can be a drawback with small learning sets, because, in this case, it may tend to over generalize, whereas SMART+ is easier to control.

4.2. Artificial Character Recognition

A second case study concerns SMART+ only, and is focused on the algorithms for learning continuous intervals. More specifically, it shows how the different algorithms described in Sections 3.2 and 3.3 cooperate, producing a very accurate knowledge base.

This domain, even though artificial, is quite complex and bears many features of a real world one. It has been described in (Botta & Giordana, 1993), in order to illustrate the learning strategies of SMART+.

Ten capital letters of the English alphabet have been chosen, and have been described as they could appear after segmenting the patterns produced by an electric pen. Each letter is composed of a set of segments that are described by the initial and final (x, y) coordinates in a Cartesian plane. From these attributes, other properties can be extracted, such as the length of a segment, its orientation, its preceding and following segments, and so on. Some of these features are numeric in nature, and then the capacity of learning and refining continuous intervals can be tested.

A dataset of 6000 examples (600 for each letter) has been generated using a program that introduces random noise, simulating segmentation errors (insertion and deletion) and variability in the segment size with respect to a basic prototype. The dataset has been divided into a balanced learning set of 1000 examples (100 for each letter) and a test set of 5000 examples. Then SMART+ has been run several times under different conditions:

1. Using only its inductive and deductive strategies, without making use of its capability of learning internal disjunctions. In order to cope with numeric features, predicates containing an internal disjunction, handcrafted by a human expert, have been used. Moreover, SMART+ was guided by a domain theory describing some typical structural features of each letter.

2. Using the symbolic inductive and deductive strategies guided by a domain theory and making use of *Algorithm 2* of Section 3.1 for learning continuous intervals.

3. Using the Genetic Algorithm-based postprocessor for refining a learned knowledge base, using the algorithm described in Section 3.3.

The results are summarized in Table 1. The relevance of the possibility of learning numeric features is evident. In fact, results obtained using only the symbolic learning component are poor, whereas the local learning strategy is sufficient to raise the recognition rate above 98%. Finally, the effect of the global GA optimization is remarkable, not only because the classification rate goes close to 100%, but also because the number of formulas necessary for the classification is almost halved. This is again due to the generalization power of the genetic component, which, as already noticed for REGAL, was able to contextually optimize the internal disjunctions, so that the number of instances covered by each rule doubled, on average. In this way many rules became redundant and could be dropped.

Table 1. Results obtained by SMART+ on the test set of the artificial character recognition case study

Strategy	N. of Formulas	Classification Rate
Pure Symbolic	81	83.44
Symbolic + Algorithm 2	47	98.68
Symbolic + Algorithm 2 + GA	28	99.7

4.3. Gradient Descent Optimization Method

This third case study concerns SNAP, and demonstrates the method for combining the symbolic learning in FOL with the quadratic error gradient descent. The task is prediction of the Mackey-Glass chaotic series, a standard benchmark. Numerical series prediction is a regression task, but it can be reduced to a classification task by discretizing the domain of the target function and considering each discrete value as a class. This approach has been adopted in (Sammut et al., 1992) to learn a flight controller for a simulator. More recently, a similar method has been proposed in (Baroglio et al., 1996) to synthesize fuzzy controllers. Here we adopted the same approach in combination with the methodology described in Section 3.4 for tuning numeric intervals.

The Mackey-Glass domain has been divided into 12 equal intervals. Then, a set of classification rules has been learned for each one of the 12 classes. The resulting classification theory has been transformed into a RBFN according to the procedure described in Section 3.4. The only variant, which is task specific, is the way the output value is constructed; in fact, the twelve outputs must be merged in order to construct a single output corresponding to the value of the function to predict. For this reason, all the Radial Basis Function units have been connected to the same output, which acts as a multiplexer. The weight on every connection has been set equal to the value of the class the corresponding RBF is predicting. Afterwards the RBFN has been refined for 5000 epochs using the Δ-rule, in order to follow the quadratic error gradient descent.

The Mackey-Glass chaotic series is generated by the following differential equation:

$$\dot{x} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{17}$$

with $x(t<0)=0$, $x(0)=1.2$ and $\tau=17$. The classical learning problem is defined as follows: given the series values at instants t, t-6, t-12, t-18, forecast the series value at t+84. Table 2 shows the results of two experiments, in terms of the Non-Dimensional Error Index (NDEI), i.e., the ratio between the average square root error and the standard deviation.

The first two lines reports the result obtained with CART, reported in (Baroglio et al., 1996), and the result obtained by (Moody & Darken, 1988) using a RBFN constructed with an enhanced version of the k-means algorithm. In the first experiment, SNAP was provided with predicates containing a single variable only, and the same features used in the literature were used for learning a RBFN, so that the induction task was actually reduced to a propositional case. The result was a network performing as well as classical RBFNs, but obtained with a learning set 10 times smaller.

Table 2. Results on the Mackey-Glass series learning problem on a test set of 500 instances.

Method	N. of Learning Instances	NDEI
$CART$	1000	0.36
$RBFN_{k-means}$	10000	0.055
$RBFN_{SNAPprop}$	1000	0.056
$RBFN_{SNAPfo}$	1000	**0.024**

In the second experiment, we used a First Order representation of the instances: each example consists of 19 objects described by two attributes, the relative position inside a time window and the corresponding series value at that point. In this second case, we defined only two predicate templates, which refer to the series value and the difference between the values at instants t-x and t-y, (x≠y∈[0,18]), respectively.

As it can be seen, the First Order Logic representation leads to substantially better performances than the propositional counterpart. It should be noted that the used First Order representation language allowed SNAP to analyze a large number of possible features (190) and to select the most relevant ones (45 on average), resulting in very good network layouts.

The last consideration is about the computational effort for learning the network layout: Table 3 reports the number of formulas searched by SNAP per class in the propositional and First Order settings. It should be pointed out that only for the First Order setting all these values were largely smaller than the computational resources allocated for such tasks (500 formulas per class).

Table 3. Computational effort on the Mackey-Glass case study.

Class	1	2	3	4	5	6	7	8	9	10	11	12
$SNAP_{prop}$	18	339	500	313	256	500	459	500	487	500	420	250
$SNAP_{fo}$	26	77	120	80	118	130	233	127	176	139	136	144

4.4. Other Examples

Other examples of applications are reported in this section, to provide the reader with a feeling of the application range available to the proposed framework. In the chess domain, the chess-endgame KR-KR problem was one of the first handled by an early REGAL prototype (Giordana & Sale, 1992). Using a learning set of 1000 examples, REGAL learned the concept of *safe black king* with an accuracy of 99.75%, evaluated on a test set of 2000 instances.

The King-Rook-King end-game problem (Quinlan, 1983) has been discussed in (Botta, 1994). Using 160 learning examples (scenarios) and 1000 test examples, a simplified version of SMART+ was able to reach 99.4% accuracy without any a-priori knowledge. As a comparison, FOIL required 1000 examples to reach the same accuracy. By adding a domain theory, it was able to reach 100% accuracy.

Another case study, designed explicitly for challenging REGAL, is the *Thousand Trains* dataset, extending the well known train set used by Michalski (Michalski, 1983). There are two concepts to discriminate: “Trains going East” and “Trains going West”. Thousands of trains have been generated by a program which selects at random the values of the attributes. Then, each train has been classified using a set of disjunctive rules. The challenge for REGAL was to discover the original rules or a set of equivalent ones. The formulation of this problem dates to (Giordana & Saitta, 1993) and the current best results obtained by REGAL have been published in (Giordana & Saitta, 1994). The concept description language used by REGAL is very similar to the one described in (Michalski, 1983). REGAL found 5 disjuncts covering completely and consistently both the learning set and the test set. This application, being available to the ML community, could be a good test bed for other relational systems, because it offers much greater challenges, given the number of instances to cover, than the now popular 20-trains dataset.

Other test-beds for REGAL have been the *Mushrooms* and the *Splice Junctions* datasets from Irvine repository. The obtained results are reported in (Neri & Saitta, 1996). A successful application of Enigma, the direct ancestor of SMART+ that was already provided with an earlier version of the algorithm for learning continuous intervals, was the automated generation of a troubleshooter (Giordana et al., 1993b). This represented a real application, in industry, of a learning system based on FOL. In fact, the troubleshooter was used for several years in field for maintenance and training purposes. The success of Enigma was in part due to the possibility of easily integrating it with a commercial relational database used to record troubleshooting sessions performed by a human expert; also this dataset could be a hard test-bed for other FOL learning algorithms.

5. Discussion

In the previous sections we have presented a framework for integrating multiple learning strategies, which is characterized by two basic choices:
(a) a separation between the (logical) hypothesis description language, and the (object oriented) instance description language,
(b) the adoption of the annotated predicate calculus, as hypothesis description language, with internal disjunction as basic construct.

It is worth noting that the two choices are independent, even though they nicely combine together. On the other hand, the first choice is, to a large extent, motivated by the need of accessing data in large databases. The representation framework, as it has been described in this paper, is not comprehensive of all the facilities that it can supply; in fact, the description only presents those features that can be used at the same time by a multistrategy approach including the symbolic, genetic and connectionist paradigms. More specifically, the biases of the core framework, and the possible extensions, can be summarized in the following. A brief comparison with alternative approaches is also sketched.

Instance Description Language (IDL) - Instances are represented as tuples (objects with attributes) in a database. Properties can be defined on the objects either by means of semantic functions or directly by providing the corresponding relations in extensional form.

In many ILP systems, instances as represented as ground clauses (facts), and this is true also for Induce-like learners. FOIL represents instances as tuples in a table (Quinlan, 1990). A comment is in order about the semantic setting. In the normal ILP setting, hypotheses are only required to be consistent with the negative instances, so that what is not explicitly stated as true is unknown, and hypotheses covering unknown facts can be accepted. In the non-monotonic semantic setting (De Raedt & Džeroski, 1994), instead, the Closed World Assumption (Clark, 1978) is done, and, hence, what is not declared true, is false. As a consequence, in the non-monotonic setting the space of acceptable hypotheses is more constrained. In our framework we also accept the same assumption and, then, we work in the same setting as (De Raedt & Džeroski, 1994). Also, in their approach, each example is a separate *mini-database* (a finite interpretation of a clausal theory), similar to our *scenario*.

Hypothesis Description Language (HDL) - A set of operational predicates (Mitchell et al., 1986) are supplied. They can be constraint predicates, with no internal disjunction, or learnable predicates, with one term in disjunctive form. The semantic of the predicates is defined in terms of properties, and negation of atoms is allowed. A well-formed HDL formula is any range-restricted, non recursive, normal clause. Starting from this core, both stronger and weaker biases can be used, depending on the specific learning task. For instance, in the system RTL (Giordana et al., 1993a, Baroglio & Botta, 1995) stratified recursive theories (Apt et al., 1988) can be learned, and in SMART+ existential and numerical quantification (Michalski, 1983) is allowed. On the other hand, a form of determinacy (extensionally controlled) can be imposed for limiting the growth of too large relations. Finally, new predicates and terms can be built up through an abstraction mechanisms in SMART+ (Giordana & Saitta, 1990, Giordana et al., 1991).

Most FOL concept learners use a form of logical language for representing hypotheses. In (Ade et al., 1995) a language used to describe biases in general, and in ILP systems in particular, is proposed. Language biases of ILP systems are rather variables, including normality, ij-determinacy, range-restriction, limitation on the number of variable, and so on. Most of them allow for recursion and for term construction, such as the one emerging in an *append* of an atom to a list.

Background Knowledge (BK) - In the presented core framework there is no background knowledge, at least formally. Actually, object properties, represented as semantic mappings, are a kind of background knowledge; they could be represented as Horn clauses and used deductively. On the other hand, complex mechanisms to handle possibly large bodies of domain theories have been implemented and used, integrating induction and deduction, both in (Bergadano & Giordana, 1988) and in SMART+. Moreover, in the system WHY, also a causal model of the domain can be used in an abuctive way (Baroglio et al., 1994, Saitta et al., 1993). Finally, a domain theory can be used to deduce the language template for REGAL (Giordana & Saitta, 1994, Giordana & Neri, 1996).

Following the proposal of using an Explanation-Based approach to learning (Mitchell et al., 1986, DeJong & Mooney, 1986), the ability to handle background knowledge has been incorporated in several systems (only as an example, we can mention FOCL (Pazzani & Kibler, 1992)). In some ILP systems the background knowledge is nothing else than a list of known facts, i.e., ground literals; in others, a background knowledge, expressed in clausal form, can also be exploited.

Search Methods - In the basic framework, several search methods and strategies can be applied. In particular, inductive hill-climbing, best-first, depth-first and beam search, controlled by a variety of heuristic evaluations (Botta & Giordana, 1993), can be used in SMART+. Genetic search is used in REGAL, and backpropagation or Δ-rule are used in SNAP (Botta & Giordana, 1996). All the above search strategies have been used in other learning systems. Most ILP systems use hill-climbing, but also beam search (Pompe & Kononenko, 1995) and bi-directional search (Muggleton, 1995) have been used.

As a first conclusion, we would like to stress that the core framework, beyond offering time and space efficiency and lending itself to be exploited by different paradigms, proved to be sufficiently powerful to be applied to significant learning tasks (Neri & Saitta, 1996, Baroglio et al., 1996).

By analyzing the above outlined commonalties and differences between alternative approaches, we believe that there are two fundamental differences: one is in the representation of instances as objects in a database, and the other is the extensive use of internal disjunction. Keeping separate by a functional abstraction layer the extension (database) from the intension (learned knowledge) offers the advantage that efficient mechanisms for managing extensional relations can be designed by exploiting the database technology. The method becomes particularly effective when object properties have a closed-form definition: in this case most of the intermediate relations, necessary to compute the extension of a hypothesis, do not need to be created; they can be obtained, instead, on demand by evaluating the properties of objects bound to the variables.

Another benefit is the flexibility at the application level. When the user wants to introduce or delete a feature, it is not necessary to do any pre-processing on the dataset, but it is sufficient to define/delete the properties necessary to compute the new feature. New properties can also be defined on-line by the learning system itself, if a constructive learning strategy is used. In this way, more robust learners can be designed. On the other hand, this approach has the drawback that the constants defined in the dataset cannot directly occur in the logical formulas, but they have to be manipulated through suitable objects. Moreover, single instances cannot be accessed directly, but only through the general operators which are applicable to the relations. These problems already emerged in the deductive database field (Minker, 1988, Vieille, 1986, Ullman, 1982, Gardin & Simon, 1987, Henschen & Naqvi, 1984), where solutions have been proposed. However, even taking advantage of such solutions, it is true that the presented framework may be quite inefficient when one must work not on whole relations but on single items, such as constants or instances.

On the contrary, the Logic Programming framework, adopted in ILP, takes the opposite point of view, and the basic operation (unification) works on single bindings and does not make distinction between ground and non ground assertions. The advantage is flexibility in handling every kind of terms, including constants. The price paid is a greater inefficiency in handling very large extensions. Therefore, going towards very large datasets, as required by KDD challenge, we have to expect that the basic ILP framework would be inadequate. In fact, recent papers describe solutions for interfacing ILP systems to relational databases by mapping logical formulas onto SQL programs (Brockhausen & Morik, 1996).

The second basic difference, namely the extensive use of internal disjunction, is not at the level of language bias, because any formula containing internal disjunctions can always be translated into a set of standard clauses. The difference emerges at the level of the very learning methodology, because this construct requires specific algorithms and generalization/specialization operators to be dealt with. The internal disjunction allows both more compact hypotheses to be found and also more reliable ones, because each hypothesis is supported by a larger coverage of the instances, not subdivided into smaller sets.

Some additional differences can be pointed out. However, these differences do not concern facilities which are or are not possible in one or the other framework, but, rather, which may be realized in each of them with greater or less effort. One is the easiness of implementing various search strategies even inside the syntactic paradigm. As in our framework the search tree is stored in a database, the search focus can be moved from one node to another in the frontier without the need of saving the current state of the search. This feature is not immediately available in the standard ILP framework, so that developing search strategies different from hill climbing may be more costly. The second is that continuous-valued attributes can be handled, in our framework, in a very natural way, and numerical intervals and discrete value sets can be made transparent to the user; again, this effect derives from the use of internal disjunction.

6. Conclusions

The first contribution of this paper has been to show that, in order to achieve effectiveness in demanding learning tasks, the abstract specification of algorithms is not sufficient, and that a good underlying engineering work is essential to obtain success. In particular, we have shown that choosing an implementation platform based on database technology offers an excellent support to learn in structured domains. Several among the described methodological aspects are not new; for example, internal disjunction has been introduced by Michalski (Michalski, 1983, Michalski & Chilausky, 1980). The idea of introducing a semantic gap between the concept description language and the instance representation derives from EBL (Mitchell et al., 1986); in fact, the functional layer, which abstracts the object properties from the basic descriptions, can be seen as a domain theory that makes operational the predicates defined in the concept description language. However, the way in which these ideas are realized, inside the described unifying framework, makes the difference, and transforms an interesting algorithm into a system capable of accepting the challenge of KDD applications. The proposed framework offers several benefits, not immediately available in other approaches:

- The database technology allows very large datasets, as they are usually found in the archives, to be handled without any need of format conversion.

- The possibility of using SQL or other languages for “real programmers” makes it easier to bring a machine learning tool inside a company (Giordana et al., 1993b).

- The functional layer, filling the semantic gap between logics and data, offers an abstraction mechanism, which can be exploited by multistrategy learners integrating heterogeneous algorithms, possibly implemented in different languages. The algorithms for learning internal disjunctions and dealing with uncertainty have been implemented in this layer, and they can cooperate due to the common interface towards the data level.

- The definition of object properties by means of functions allows one to define a great number of virtual properties, computed on demand without the need of creating their extension since the beginning. This feature is widely exploited, for instance, by the algorithms learning continuous intervals.
- A last benefit, not described here, is the possibility of defining new features, or constructing new objects, as it is done in a constructive learning approach (Wnek & Michalski, 1994).

A more technical contribution of the paper resides in the algorithms for learning internal disjunctions. Besides their value per se, such algorithms show how, in practice, it is possible to combine into a unique multistrategy framework three different paradigms, namely the symbolic, the genetic and the connectionist one, exploiting the best from each of them. This result widens the perspectives of developing multistrategy approaches for learning in FOL. Finally, the validity of a multistrategy approach has been confirmed by the results presented in Section 4, which have been obtained only by virtue of the cooperation among several algorithms, whose peculiarities have been synergistically exploited.

Notes

1. Here a notation in the style of standard object oriented programming languages has been used. Commercially available relational and object-oriented databases offer the support for implementing the dataset representation described so far, so that instance structure and object properties can be described in a SQL-like language.
2. Notice that, in this way, the object properties necessary to the selection operator are computed on demand, exiting so from the scheme of pure relational algebra (Ullman, 1982).
3. We remember that a model of a theory is a binding which verifies the theory.

References

Ade, H., De Raedt, L., & Bruynooghe, M. (1995). Declarative bias for specific-to-general ILP systems. *Machine Learning*, 20:119–154.

Apt, K., Blair, H., & Walker, A. (1988). Towards a theory of declarative knowledge. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA.

Bala, J., De Jong, K., & Pachowicz, P. (1991). Learning noise tolerant classification procedures by integrating inductive learning and genetic algorithms. In *First International Workshop on Multistrategy Learning*, pages 316–323, Harpers Ferry, WV.

Baroglio, C. & Botta, M. (1995). Multiple predicate learning with RTL. In *4th Congress of the Italian Association for Artificial Intelligence AI*IA'95*, LNAI-992, pages 44–55, Florence, Italy.

Baroglio, C., Botta, M., & Saitta, L. (1994). WHY: A system that learns using causal models and examples. In Michalski, R. and Tecuci, G., editors, *Machine Learning: A Multistrategy Approach*, volume IV, pages 319–347. Morgan Kaufmann, San Francisco, CA.

Baroglio, C., Giordana, A., Kaiser, M., Nuttin, M., & Piola, R. (1996). Learning controllers for industrial robots. *Machine Learning*, 23:221–250.

Bäck, T. (1995). Generalized convergence models for tournament- and (μ, λ) -selection. In *6th International Conference on Genetic Algorithms*, pages 2–8, Pittsburg, PA.

Berenji, H. (1992). An architecture for designing fuzzy controllers with neural networks. *International Journal of Approximate Reasoning*, 6(2):267–292.

Bergadano, F. & Giordana, A. (1988). A knowledge intensive approach to concept induction. In *Proceedings of the 5th International Conference on Machine Learning*, pages 305–317, Ann Arbor, MI. Morgan Kauffman.

Bergadano, F., Giordana, A., & Saitta, L. (1988). Learning concepts in noisy environment. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, PAMI-10:555–578.

Bergadano, F., Giordana, A., & Saitta, L. (1991). *Machine Learning: An Integrated Framework and its Applications*. Hellis Horwood, Chichester, UK.

Blanzieri, E. & Katenkamp, P. (1996). Learning radial basis function networks on-line. In *13th International Conference on Machine Learning*, pages 37–45, Bari, Italy.

Botta, M. (1994). Learning first order theories. In *8th International Symposium on Methodologies for Intelligent Systems*, LNAI-869, pages 356–365, Charlotte, NC.

Botta, M. & Giordana, A. (1993). SMART+: A multi-strategy learning tool. In *IJCAI-93, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 937–943, Chambéry, France.

Botta, M. & Giordana, A. (1996). Combining symbolic and numeric methods for learning to predict temporal series. In *Proceedings of the 3rd International Workshop on Multistrategy Learning*, pages 234–249, Harpers Ferry, WV.

Botta, M., Saitta, L., Brancadori, F., De Marchi, D., & Radicchi, S. (1992). Automatic construction of second generation diagnostic expert systems. *International Journal of Expert Systems*, 4:389–400.

Bratko, I. & Džeroski, S. (1995). Engineering applications of ILP. *New Generation Computing*, 13:313–333.

Brockhausen, P. & Morik, K. (1996). Direct access of an ILP algorithm to a database management system. In *Proc. of the MLnet Familiarization Workshop*, pages 95–110, Bari, Italy.

Buntine, W. (1988). Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36:149–176.

Cameron-Jones, R. & Quinlan, R. (1993). Avoiding pitfalls when learning recursive theories. In *IJCAI-93, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1050–1055, Chambéry, France.

Clark, K. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, NY.

De Jong, K. A., Spears, W. M., & Gordon, F. D. (1993). Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188.

De Raedt, L. & Džeroski, S. (1994). First-order jk-clausal theories are PAC -learnable. *Artificial Intelligence*, 70:375–392.

DeJong, G. & Mooney, R. (1986). Explanation based learning: an alternative view. *Machine Learning*, 1:145–176.

Dietterich, T. & Michalski, R. (1983). A comparative review of selected methods for learning from examples. In Carbonell, J., Michalski, R., and Mitchell, T., editors, *Machine Learning, an Artificial Intelligence Approach*, pages 41-82. Morgan Kaufmann, Los Altos, CA.

Faure, C., Frediani, S., & Saitta, L. (1993). A semiautomated methodology for knowledge elicitation. *IEEE Transaction on Systems, Man, and Cybernetics*, SMC-23:346–356.

Flach, P. (1995). *Conjectures: An Inquiry Concerning the Logic of Induction*. PhD thesis, Tilburg University (Netherlands), CIP-Gegevens KoninKlijke Bibliotheek, Den Haag.

Gardin, G. & Simon, E. (1987). Les systèmes de gestion de bases des données déductives. *Synthèse*, 6.

Gemello, R., Mana, F., & Saitta, L. (1991). Rigel: An inductive learning system. *Machine Learning*, 6:7–36.

Giordana, A. & Neri, F. (1996). Search-intensive concept induction. *Evolutionary Computation*, 3:375–416.

Giordana, A. & Saitta, L. (1990). Abstraction: a general framework for learning. In *Working Notes of the AGAA-90 Workshop*, pages 245–256, Boston, MA.

Giordana, A. & Saitta, L. (1993). REGAL: an integrated system for learning relations using genetic algorithms. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 234–249, Harpers Ferry, WV.

Giordana, A. & Saitta, L. (1994). Learning disjunctive concepts by means of genetic algorithms. In *Proceedings of the 11th International Conference on Machine Learning*, pages 96–104, New Brunswick, NJ.

Giordana, A., Saitta, L., & Baroglio, C. (1993a). Learning simple recursive theories. In *Methodologies for Intelligent Systems, Proc. of the 7th International Symposium, ISMIS-93*, pages 425-434, Trondheim, Norway. Springer-Verlag.

Giordana, A., Saitta, L., Bergadano, F., Braucadon, F., & De Marchi, D. (1993b). Enigma: a system that learns diagnostic knowledge. *IEEE Transactions on Knowledge and Data Engineering*, 5(1):15–28.

Giordana, A., Saitta, L., & Roverso, D. (1991). Abstracting concepts with inverse resolution. In *Proc. of 8th International Workshop on Machine Learning*, pages 142–146, Evanston, IL.

Giordana, A. & Sale, C. (1992). Genetic algorithms for learning relations. In *Proc. of 9th International Conference on Machine Learning*, pages 169–178, Aberdeen, UK.

Goldberg, D. (1989). *Genetic Algorithms*. Addison-Wesley.

Gordon, D. & desJardins, M. (1995). Evaluation and selection of biases in machine learning. *Machine Learning*, 20:5–22.

Greene, D. & Smith, S. (1993). Competition-based induction of decision models from examples. *Machine Learning*, 13:229–258.

Grefenstette, J., Ramsey, C., & Schultz, A. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5:355–381.

Haussler, D. (1988). Learning conjunctive concepts in structural domains. *Machine Learning*, 4:7–40.

Hayes-Roth, F. & McDermott, J. (1978). An interference matching technique for inducing abstractions. *Communications of the ACM*, 21:401–411.

Helft, N. (1989). Induction as nonmonotonic inference. In *Proc. of 1st Conference on Knowledge Representation and Reasoning*, pages 149–156, Boston, MA.

Henschen, L. J. & Naqvi, S. A. (1984). On compiling queries in recursive, first order databases. *Journal of ACM*, 31, pages 47-85.

Janikow, C. (1993). A knowledge intensive genetic algorithm for supervised learning. *Machine Learning*, 13:198–228.

Kodratoff, Y. & Ganascia, J. (1986). Improving the generalization step in learning. In Carbonell, J.G., Michalski, R., and Mitchell, T. M., editors, *Machine Learning, an Artificial Intelligence Approach*, volume II, pages 215–244. Morgan Kaufmann, San Mateo, CA.

Lavrač, N., Džeroski, S., & Grobelnik, M. (1991). Learning nonrecursive definitions of relations with LINUS. In *Proceedings of the 5th European Working Session on Learning*, pages 265–281, Porto, Portugal.

McCallum, R. & Spackman, K. (1990). Using genetic algorithm to learn disjunctive rules from examples. In *Proc. of the 7th International Conference on Machine Learning*, pages 149–152, Austin, Texas.

Michalski, R. (1980). Pattern recognition as a rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2:349–361.

Michalski, R. (1983). A theory and methodology of inductive learning. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134, Los Altos, CA. Morgan Kaufmann.

Michalski, R. (1991). Inferential learning theory as a basis for multistrategy task-adaptive learning. In *Proc of 1st International Workshop on Multistrategy Learning*, pages 3–18, Harpers Ferry, WV.

Michalski, R. & Chilausky, R. (1980). Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, 4:125–126.

Minker, J., editor (1988). *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, Los Altos, CA.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation based generalization: a unifying view. *Machine Learning*, 1:47–80.

Moody, J. & Darken, C. (1988). Learning with localized receptive fields. In Sejnowski, T., Touretzky, D., and Hinton, G., editors, *Connectionist Models Summer School*, Carnegie Mellon University.

Morik, K. (1991). Balanced cooperative modeling. In *Proc. of the 1st International Workshop on Multistrategy Learning*, pages 65–80, Harpers Ferry, WV.

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8:295–318.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, 13:245–286.

Muggleton, S. & Feng, C. (1990). Efficient induction of logic programs. In *Proc. of the 1st Conference on Algorithmic Learning Theory*, pages 368–381, Japan.

Neri, F. (1997). First order Logic Concept Learning by means of a Distributed Genetic Algorithms. Phd. Thesis, University of Torino, Italy, available at `http://www.di.unito.it.ct/~neri/phd/thesis.ps.gz`

Neri, F. & Saitta, L. (1996). Exploring the power of genetic search in learning symbolic classifiers. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-18:1135–1142.

Opitz, D. & Shavlik, J. (1993). Heuristically expanding knowledge-based neural networks. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1360–1365, Chambéry, France.

Opitz, D. & Shavlik, J. (1995). Dynamically adding symbolically meaningful nodes to knowledge-based neural networks. *Knowledge Based Systems*, 8:301–311.

Pagallo, G. (1989). Learning DNF by decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 639–644, Detroit, MI.

Pazzani, M. & Kibler, D. (1992). The utility of knowledge in inductive learning. *Machine Learning*, 9:57–94.

Plotkin, G. (1970). A note in inductive generalization. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, volume V, pages 153–163.

Poggio, T. & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497.

Pompe, U. & Kononenko, I. (1995). Linear space induction in first order logic with RELIEF. In *ISSEK Workshop*, volume CISM Lecture Notes. Springer-Verlag.

Quinlan, R. (1983). Efficient classification procedures. In Carbonell, J., Michalski, R., and Mitchell, T., editors, *Machine Learning, an Artificial Intelligence Approach*, pages 463-482. Morgan Kaufmann, Los Altos, CA.

Quinlan, R. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239–266.

Rumelhart, D., Hinton, G., & Williams, R. (1985). Learning internal representations by error propagation. Technical Report 8506, Institute for Cognitive Science, La Jolla: University of California, San Diego.

Rumelhart, D. E. & McClelland, J. L. (1986). *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Parts I & II.* MIT Press, Cambridge, Massachusetts.

Saitta, L., Botta, M., & Neri, F. (1993). Multistrategy learning and theory revision. *Machine Learning*, 11:153–172.

Saitta, L., Giordana, A., & Neri, F. (1995). What is the ”Real World”? In *Proc. of Workshop on Applying Machine Learning in Practice*, pages 34–40, Lake Tahoe, CA.

Sammut, C., Hurst, S., Kedzier, D., & Michie, D. (1992). Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning (ML92)*, pages 385–393. Morgan Kaufmann.

Shapiro, E. (1983). *Algorithmic Program Debugging*. The MIT Press, Cambridge, MA.

Silverstein, G. & Pazzani, M. (1991). Relational cliches: Constraining constructive induction during relational learning. In *Proc. of the 8th International Workshop on Machine Learning*, pages 203-207, Evanston, IL.

Towell, G., Shavlik, J., & Noordwier, M. (1990). Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the* 8^{th} *National Conference on Artificial Intelligence AAAI'90*, pages 861–866.

Tresp, V., Hollatz, J., & Ahmad, S. (1993). Network structuring and training using rule-based knowledge. In *Advances in Neural Information Processing Systems 5 (NIPS-5)*.

Ullman, J. (1982). *Principles of Databases*. Computer Science, Baltimore, MD.

Vafaie, H. & De Jong, K. (1991). Improving the performance of rule induction system using genetic algorithms. In *Proc. of 1st International Workshop on Multistrategy Learning*, pages 305–315, Harpers Ferry, WV.

Vere, S. (1978). Inductive learning of relational production. In *Pattern-Directed Inference System*, pages 281–295. Academic Press, London, UK.

Vieille, L. (1986). Recursive axioms in deductive databases: the query subquery approach. In *Proc. of First International Conference on Expert Database Systems*, Charleston, SC.

Winston, P. (1975). Learning structural descriptions from example. In Winston, P., editor, *The psychology of computer vision*, pages 157–209. McGraw Hill, New York, NY.

Wnek, J. & Michalski, R. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, 14(2):139–168.

Zadeh, L. (1965). Fuzzy sets. *Information Control*, 8:338–353.

Zadeh, L. (1992). Knowledge representation in fuzzy logic. In Yager, R. and Zadeh, L., editors, *An Introduction to Fuzzy Logic Applications in Intelligent Systems*. Kluver Academic Publishers.

Received Setemper 3, 1996
Accepted December 13, 1996
Final Manuscript January 28, 1997