How to Upgrade Propositional Learners to First Order Logic: a Case Study

Wim Van Laer (1) and Luc De Raedt (2)

(1) Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
Email: `Wim.VanLaer@cs.kuleuven.ac.be`
(2) Institut für Informatik, Albert-Ludwigs-Universität Freiburg
Am Flughafen 17, D-79110 Freiburg, Germany
Email: `deraedt@informatik.uni-freiburg.de`

June 8, 2000

Abstract

We describe a methodology for upgrading existing attribute value learners towards first order logic. This method has several advantages: one can profit from existing research on propositional learners (and inherit its efficiency and effectiveness), relational learners (and inherit its expressiveness) and PAC-learning (and inherit its theoretical basis). Moreover there is a clear relationship between the new relational system and its propositional counterpart. This makes the ILP system easy to use and understand by users familiar with the propositional counterpart. We demonstrate the methodology on the ICL system which is an upgrade of the propositional learner CN2.

Keywords: propositional learning, relational learning, learning from interpretations, inductive logic programming, classification.

1 Introduction

Current machine learning systems are often distinguished on the basis of their representation, which can either be propositional or first order logic. Systems belonging to the first category are often called attribute value learners, systems of the second category are called relational learners or inductive logic programming systems.

In this paper, we shall argue that it is effective to develop first order learners that have existing propositional machine learning systems as a special case. Advantages include: ease of understanding and use of the first order system by users familiar with the propositional case; potential exploitation of all expertise (and heuristics) available for the propositional learner; a clear relation between the propositional learner and its first order variant, resulting in e.g. identical results on identical (propositional) data.

Given this viewpoint we develop a methodology for upgrading propositional learners towards first order logic and demonstrate it at work. This methodology is perhaps the most important lesson learned during the development of several inductive logic programming systems and results (including [21], Tilde [9, 7], ICL [23], Claudien [20], Warmr [26]) of the machine learning group in Leuven. The methodology starts from an existing propositional learner and provides a recipe for upgrading it towards the use of first order logic. The recipe involves the use of examples which correspond to sets of ground facts (interpretations), the adaptation of the representation of hypotheses towards Prolog, the employment of θ-subsumption to structure and search the space of hypotheses, the introduction of a declarative bias, and otherwise recycles as much as possible from the original system. Following the methodology, it should be easy to turn virtually any propositional symbolic learner into an inductive logic programming system.

To show how the methodology works, we demonstrate it on upgrading the well-known CN2 [14, 13] learning algorithm towards ICL [23]. In Section 6 we give an overview of other systems that follow the same methodology.

The paper is structured as follows: we first elaborate on the characteristics of the propositional and the first order knowledge representation and we show how the relational representation can overcome limitations of the propositional representation. After describing the propositional learner CN2, we present our methodology for upgrading a propositional learner and illustrate each step w.r.t. CN2 resulting in the relational learner ICL. We also present some experimental results with ICL that show that the methodology is worthwhile. In the last section we discuss some related results and conclude.

2 Knowledge Representation

2.1 Attribute value learning

Consider Figure 1. Each example or scene can be described by a fixed number of attributes: `shape-left`, `size-left`, `color-left`, `shape-right`, `size-right`, `color-right` and `class`. The data-set can be summarized in one table as in Figure 1, where each row (or in relational database terms, each tuple) represents one example. Many well-known systems like C4.5 [55, 56] and CN2 [14, 13] are based on this *attribute value representation* (also called propositional representation), and are as such called attribute value learners. Also, data mining mainly focusses on learning from single tables.

The examples in Figure 2 however, cannot easily be described by a fixed number of features. A scene or example consists of a variable number of geometrical objects (such as lines, points, squares, triangles, circles,...), each having

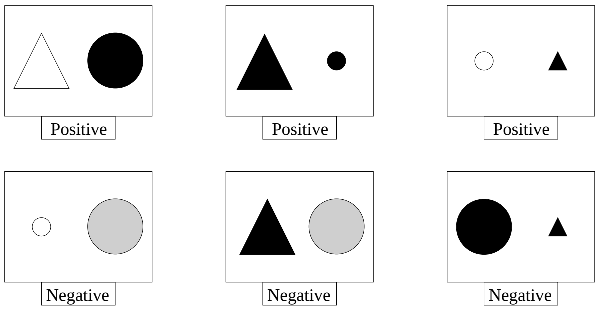

Figure 1: A simple classification problem. One scene (example) consists of a left-side and a right-side object. Each scene is tagged with a class (positive or negative).

Table 1: An attribute value representation for Figure 1 (with `id` a unique identifier for each example).

`id`	`shape-left`	`size-left`	`color-left`	`shape-right`	`size-right`	`color-right`	`class`
1	triangle	large	white	circle	large	black	positive
2	triangle	large	black	circle	small	black	positive
3	circle	small	white	triangle	small	black	positive
4	circle	small	white	circle	large	grey	negative
5	triangle	large	black	circle	large	grey	negative
6	circle	large	black	triangle	small	black	negative

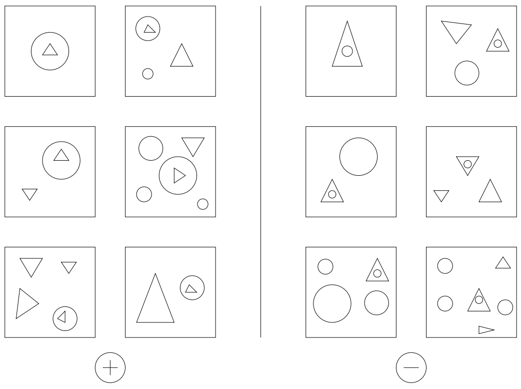

Figure 2: A more complex classification problem: Bongard problem 47, developed by the Russian scientist M. Bongard in [10]. It consists of 12 scenes (or examples), six of class $\oplus$ and six of class $\ominus$. The goal is to discriminate between the two classes.

a number of different properties (e.g., white, black, small, large, horizontal,...), and a variable number of relations between objects. Representing these scenes with a fixed set of attribute value pairs results in a number of problems (cf. [19]). *First*, one should fix the maximum number of objects in a scene. Given a bound b on the number of objects, one could then list attributes $A_{i,1}, ..., A_{i,j}$ characterizing each object i. Some of these attributes will yield nil values since not all scenes may possess the same number of objects and not all attributes/properties are meaningful for each object. *Secondly*, one should also order the objects in a scene, which is more problematic. Indeed, reconsider scene 1 in Figure 1. Its representation in Table 1 assumes that the order is from left to right. In general, the objects will be essentially unordered (as in Figure 2). Without determining the order of objects within a scene there is an exponential number of equivalent representations of a scene (in the number of objects). Scene 1 of Figure 1 corresponds to one such representation, another representation (based on a different order) would swap the left and right object. For similar reasons, the representation of rules will also require one such ordering. These two problems prohibit an efficient encoding of first order problems into the attribute value representations employed by typical machine learning systems. *Thirdly*, one should provide an attribute for each possible relation between each pair of objects (in a specific order). E.g. the first object is left of the second object, the first object is left of the third object,... Again, the number of such attributes (relations) grows exponentially in the number of objects available.

Though the above problem is a toy-problem, it is very similar to real-life problems in e.g. the field of molecular biology (see [42, 11]) where essentially the same representational problems arise. Data consists of a set of molecules, each of which is composed of several atoms with specific properties (like charge). Similar to a scene, there exists a number of relations between atoms (like bonds, structures,...).

2.2 First order representations

The above sketched problems can be overcome using a *relational/first order representation*. We propose the following representation for examples:

an example is a set of ground facts

Ground facts are tuples in a relational database.

From a logical point of view this is called a (Herbrand) *interpretation* because the facts represent all atoms which are true for the example, thus all facts not in the example are assumed to be false. From a computational view this can be seen as a a small relational database or a Prolog knowledge base, so we can make use of a Prolog interpreter for querying an example.

To illustrate this representation, let's reexamine the Bongard problem in Figure 2. The upper left scene consists of a small triangle, pointing up, and which is in a large circle. This scene can be specified as follows:

{`class`(*positive*), `object`(*o1*), `object`(*o2*), `shape`(*o1*, *circle*), `size`(*o1*, *large*), `shape`(*o2*, *triangle*), `size`(*o2*, *small*), `pointing`(*o2*, *up*), `in`(*o2*, *o1*)}.

The other scenes can be encoded in the same way. The number of objects in one scene is not limited, and objects are not ordered (they could be called a and b instead of $o1$ and $o2$). Different objects can have different properties, e.g. a triangle can be pointing up or down, but this property makes no sense for a circle. And finally, the number of relations between objects is unlimited.

This first order representation is more general and more expressive than the attribute value representation which is a special case of it. Indeed, an attribute value table with k attributes can be mapped to a set of interpretations/examples as follows:

> For each example (a tuple/row in the attribute value table), construct the fact {example(val_1, ..., val_k).}, where val_i is the value of the ith attribute of the example in the table. Then each of these facts is the interpretation of the corresponding example.
>
> Instead of mapping to one fact, an alternative is to map each row or example to a set of k facts {$\text{att}_1(val_1)$, ..., $\text{att}_k(val_k)$} where val_i is the value for the ith attribute.

For instance, the first example in Figure 1 can be represented by

{`example`(*triangle*, *large*, *white*, *circle*, *large*, *black*, *positive*)}

or

{`shape-left`(*triangle*), `size-left`(*large*), `color-left`(*white*), `shape-right`(*circle*), `size-right`(*large*), `color-right`(*black*), `class`(*positive*)}

At this point, it is worth noting that in the attribute value representation each example must have a single value for a given attribute. Therefore, if we know that the value of e.g. `size-left`=*large*, we also know that `size-left`$\neq$*small*. This corresponds to making a kind of closed world assumption at the level of each example (cf. [37]). Due to the use of Prolog (and the implicit negation as failure), the meaning of each example in the above representation is correctly captured! (i.e. if `color-left`(*black*) then ?-`color-left`(X) will only succeed for X=*black*.)

This framework and use of Prolog is quite similar to what happens in the older work on structural matching (e.g. [38, 39, 70, 71, 72]).

2.3 Background knowledge

It is useful to use not only factual knowledge in the examples, but also Prolog rules (or definite clauses). If these rules are common to all the examples, they are referred to as background knowledge. Such knowledge can take various forms: e.g. abstraction of specific values into a taxonomy or interval, deriving new properties from a combination of existing ones, summarizing or aggregating values of several facts/tuples into a single value, etc.

For our Bongard problem in Figure 2, we could add the following definitions:

```
polygon(X) :- triangle(X).
polygon(X) :- rectangle(X).
number_objects(NO) :- setof(O, object(O), LO), length(LO, NO).
```

The first two clauses state that a polygon can be either a triangle or a rectangle. The last clause calculates the number of objects in an example by creating a set of all objects and counting the number of elements in this set.

As background knowledge is visible for each example, all the facts that can be derived from the background knowledge and an example are part of the extended example[1]. When querying an example, it suffices to assert the background knowledge and the example; the Prolog interpreter will do the necessary derivations.

2.4 Note

The above representation of examples is known in the literature as *learning from interpretations* [21, 18]. It is only one of the possible representations used within inductive logic programming. More details on the relation among various inductive logic programming settings can be found in [18].

3 The propositional learner CN2

CN2 is a well-known attribute value learning system which is described in [14, 13]. Originally, it induced an ordered list of rules using entropy as its search

[1]Formally, an extended example is the minimal Herbrand model of the example and the background theory.

Table 2: Learn a theory for one class.

```
Learn-For-One-Class(Examples, Class) return Hypothesis;
  1. let $P := \{e \in Examples \mid$ example $e$ is of class $Class\}$;
  2. let $N := \{e \in Examples \mid$ example $e$ is not of class $Class\}$;
  3. let $H := \emptyset$;
  4. repeat
       (a) BestRule = Find-Best-Rule(P, N);
       (b) if BestRule found then
             i. add BestRule to H;
             ii. remove from P all examples e covered by BestRule;
     until BestRule not found or P is empty;
  5. return H;
```

heuristic [14]. Two improvements to the algorithm are described in [13]: the use of the Laplace error estimate as evaluation function and the generation of unordered rules instead of ordered rules. In the rest of the paper we will only consider the algorithm for learning unordered rules.

Informally, CN2's problem specification is: given a set of (AV) examples E (represented as described in Section 2.1) and a set of classes C (each example belongs to one class), find an unordered set of rules of the form `class`=*class* if *condition* (with *condition* a conjunction of attribute-value tests) such that each example is classified correctly. To classify an example, one collects all rules which fire (i.e. all rules that cover the example[2]) and predict the class by a simple probabilistic method to resolve clashes. For the moment, we will concentrate on the task to induce a set of rules for one class c: the set of positive examples P are all the examples belonging to the class c, the set of negative examples N are all the others (so $E = P \cup N$).

Reconsider the classification problem in Figure 1. CN2 might learn the following hypothesis for class *positive* (as an unordered set of rules):

`class`=*positive* if `color-left`=*white* and `color-right`=*black*
`class`=*positive* if `size-right`=*small* and `shape-right`=*circle*

To learn a theory for one class, CN2 performs a covering approach on the positive examples: it repeatedly finds a single rule that is considered *best* (that is maximizes the positive examples covered and minimizes the negative examples covered). The *best* rule is then added to the hypothesis H and all examples of P that are covered by the rule are removed from P. This process terminates when no *best* rule can be found or when no more positives have to be covered. The algorithm can be found in Table 2.

[2]A rule covers an example if the condition of the rule is true for the example.

Table 3: Beam search algorithm to find the *best* rule.

```
Find-Best-Rule($P$,$N$);
  1. let $mgr$ := most general rule in the search space;
  2. let $Beam$ := $mgr$;
  3. let $BestRule$ := $\emptyset$;
  4. while $Beam$ is not empty do
     (a) let $NewBeam$ := $\emptyset$;
     (b) for each rule $R$ in $Beam$ do
         for each refinement $Ref$ of $R$ do
           i. if ($Ref$ is better than $BestRule$ and $Ref$ is statistically significant)
              then let $BestRule$ := $Ref$;
          ii. if $Ref$ is not to be pruned
              then
                • add $Ref$ to $NewBeam$;
                • if size of $NewBeam$ > $MaxBeamSize$
                  then remove worst rule from $NewBeam$;
     (c) let $Beam$ := $NewBeam$;
  5. return $BestRule$;
```

To find a *best* rule, CN2 has to search through the space of rules. The structure of this search space is implied by the subset test. Refining a rule is simply done by adding a new attribute test to the body of the rule (also called condition). CN2 starts with the most general rule of the search space (usually the rule with an empty body: *class* :- *true*). It then performs a beam search. At each step/level, all refinements of the rules in the current *beam* are evaluated. If the rule is statistically significant and better than the current best, it becomes the current best rule. From all the refinements, the $MaxBeamSize$ best rules are kept in the new *beam*. This search repeats until no more rules are in the *beam*. The algorithm for finding a *best* rule can be found in Table 3.

4 Upgrading CN2

In Section 2, we have motivated the need for relational representations and we have introduced a first order representation for examples. Now, we can focus on the methodology for upgrading propositional learners.

The methodology is summarized in Table 4 and discussed in detail below through a case study with CN2 [14, 13] and ICL [23]. The final section will briefly review a number of other cases with the methodology.

4.1 The propositional task and algorithm

Suppose that we are asked to design a learning system for Bongard type problems. Machine learning researchers would observe that Bongard problems are classification problems (another popular task is that of descriptive learning, for example discovering association rules [2, 1, 65]). So, the range of possible propositional learning algorithms to consider includes AQ [46], TDIDT [56] (like C4.5 [55]), and CN2 [14, 13]. Suppose we fancy the latter algorithm because it combines the advantages of AQ and TDIDT, i.e. it produces understandable rules, it is efficient and can cope with noisy data. So, we decide to base our first order learner on CN2. Then we have also accomplished the first step of the methodology:

Step 1 : *Identify the propositional learner that best matches the learning task.*

Given the goal of upgrading CN2 to first order logic, the question is how to realize this. At this point, the reader may notice that CN2 will not work on the Bongard problem because:

- the representation of the examples is propositional
- the representation of the rules is propositional
- the search operators are propositional

We will now discuss these issues in detail.

Table 4: An overview of the methodology for upgrading propositional learners to first order logic.

Step 1 *: Identify the propositional learner that best matches the learning task.*

Step 2 *: Use interpretations to represent examples.*

Step 3 *: Upgrade the representation of propositional hypotheses by replacing attribute-value tests by first order literals and modify the coverage test accordingly.*

Step 4 *: Use θ-subsumption as the framework for generality.*

Step 5 *: Use an operator under θ-subsumption. Use that one that corresponds closely to the propositional operator.*

Step 6*: Use a declarative bias mechanism to limit the search-space.*

Step 7*: Implement.*

Step 8*: Evaluate your (first order) implementation on propositional and relational data.*

Step 9*: Add interesting extra features.*

4.2 Examples are Interpretations

The propositional representation of examples should be upgraded to a first order one. We propose to use interpretations for this (see Section 2) as it is a natural representation for examples and there is a clear relation with attribute value learning [19, 18]. This will alleviate the first problem. Also, if desired, background knowledge can be formulated in Prolog as in Section 2.3.

Step 2 *: Use interpretations to represent examples.*

4.3 First order hypotheses

As the expressiveness of the examples (inputs) has been extended, we should also extend the expressiveness of the hypotheses (outputs).

Let us have a closer look at the concept representation in CN2. Recall from Section 3 that CN2 learns an unordered set of rules of the form `class`=*class* if *condition*, with *condition* a conjunction of attribute-value tests (e.g. `color-left` =*white*). An attribute-value test can be seen as a special case of a literal. For example, `color-left`=*white* can be mapped to `color-left`(*white*) (cf. also the mapping in Section 2.2). So if we allow literals (with possibly more than one argument, and with variables or terms as arguments) instead of just attribute-value tests, the hypothesis will be a kind of first order expression. When using rule sets with literals (the variables in the literals are existentially quantified), we can learn the following rule for the Bongard problem in Figure 2:

`class`=$\oplus$ if $\exists$T,C: `shape`(T, *triangle*) and `shape`(C, *circle*) and `in`(T, C).

which states that there exists a triangle and a circle (thus an instantiation for the variables T and C) such that the triangle is inside the circle. At this point, the condition of the rule corresponds to a Prolog query. Furthermore, instead of the 'if' notation in rule-based approaches it is common in logic programming and Prolog to write ':-', yielding a typical Prolog clause:

`class`($\oplus$) :- `shape`(T, *triangle*), `shape`(C, *circle*), `in`(T, C).

As a result, a first order upgrade of unordered rule sets for CN2 is of the form:

$$\begin{array}{c} \texttt{class}(\mathit{class}) \text{ :- } l_{1,1}, ..., l_{1,n_1} \\ ... \\ \texttt{class}(\mathit{class}) \text{ :- } l_{k,1}, ..., l_{k,n_k} \end{array}$$

where all $l_{i,j}$ are literals and all the variables appearing in the literals are existentially quantified. Note that the variables are independent between different rules.

So far, we have only sketched a syntactic adaptation. We also need to modify the semantics of hypotheses. This boils down to specifying when an example is covered by a hypothesis. An example e will be covered by a hypothesis H (a set of rules for `class`(c)) if $H \wedge e \models$ `class`(c). Thus to test coverage, one asserts the hypothesis H (resp. a rule) and the example e in a Prolog knowledge base (one could also use a relational database system) and runs the query ?-`class`(c). If this query succeeds, the example is covered; otherwise it is not.

Note that this coverage test is more complex in both time and space than its propositional counterpart (which is a simple subset test).

The reader familiar with CN2 may observe that CN2 also uses a simple probabilistic method to resolve conflicts/clashes when an example is covered by rules belonging to multiple classes. These probabilities can straightforwardly be used here too. It is merely the test for coverage that needs to be changed.

The third step of the methodology can be summarized as follows:

Step 3 *: Upgrade the representation of propositional hypotheses by replacing attribute-value tests by first order literals and modify the coverage test accordingly.*

This third step alleviates the second problem concerning CN2, mentioned earlier.

Note that this step also works for a wide range of propositional hypothesis/concept representations, like ordered or unordered rule sets, decision trees, regression trees, association rules,.... Indeed, all these concept descriptions have one thing in common: they are all based on attribute-value tests. For instance, in a decision tree each branch is based on an attribute-value test, and an association rule is a set of attribute-value tests.

For example, TILDE [9, 7] introduces first order logical decision trees - FOLDT (of which binary trees are a special case). A FOLDT is a binary decision tree in which the nodes of the tree contain a conjunction of literals. Moreover, different nodes may share variables, under the restriction that a variable that is introduced in a node (meaning that it does not occur in higher nodes) does not occur in the right branch of that node[3]. An example of a logical decision tree is shown in Figure 3. Note the sharing of the variable T in both literals.

4.4 Structuring the search-space

Nearly all symbolic machine learning systems structure the search-space by means of the *is more general than* relation. When working with propositional representations this relation is often quite simple. For instance, in the CN2 algorithm one rule is more general than another rule if all literals (i.e. attribute-value tests) occurring in the first rule are a subset of those occurring in the second rule.

On the other hand, when working with first order representations the frameworks for generality become rather complex. Various frameworks have been proposed, including θ-subsumption (from Plotkin [53]), inverse implication, inverse resolution and inverse entailment (cf. [49] for an overview). However, in

[3]The need for this restriction follows from the semantics of FOLDT. A variable X in a literal is existentially quantified within the conjunction of that node. As the the right subtree is only relevant when the conjunction fails (thus saying *there is no such* X), further references to X are meaningless in the right branch.

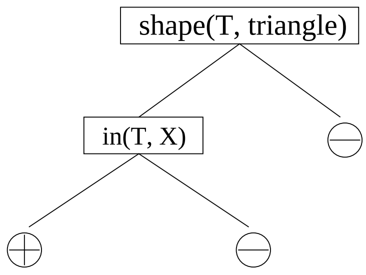

Figure 3: An example of a first order logical decision tree that discriminates between the classes $\oplus$ and $\ominus$ for the Bongard problem in Figure 2.

practice, the large majority of ILP systems (including FOIL [57], Golem [50], PROGOL[4] [48], CLAUDIEN [20], and TILDE [9, 7],...) uses θ-subsumption. This is due to the excellent computational properties of θ-subsumption (as compared to inverse resolution and inverse implication, which are both computationally intractable and less understood yet). Another important property of θ-subsumption is that it works at the level of single clauses instead of sets of clauses (as inverse resolution). This is similar to propositional systems which also structure the space at the level of individual rules. On the other hand, working at the level of single clauses only may cause problems when learning recursive clauses or multiple predicates (cf. [22, 4]). However, in our opinion, recursion is not essential for most real-life applications of relational machine learning and data mining. Indeed, the authors are only familiar of a few recursive rules in the mesh-design [27]. Most other current real-life applications of inductive logic programming do not involve recursive regularities (see [31, 11] for overviews). So, in most applications θ-subsumption is the right framework for generality when upgrading propositional systems to first order logic.

Step 4 *: Use θ-subsumption as the framework for generality.*

Before showing in the next section how to adapt the operators, we provide a brief review of θ-subsumption and its properties.

Let us first introduce the definition:

$$\text{Clause } C_1 \ \theta\text{-subsumes clause } C_2 \text{ iff } \exists\theta\text{: } C_1\theta \subseteq C_2.$$

A clauses (rule) is a set of literals and a variable substitution θ (=$\{V_1/t_1, \ldots, V_n/t_n\}$) maps each variable V_i to its corresponding term t_i. For instance:
C_1 = `father`(X, Y) :- `parent`(X, Y), `male`(X). is more general than clause C_2 = `father`(*jef*, *wim*) :- `parent`(*jef*, *wim*), `parent`(*jef*, *ann*), `male`(*jef*), `female`(*ann*). because $C_1\theta \subseteq C_2$ with $\theta = \{X/jef, Y/wim\}$.

Note that the propositional ordering on the search space is a special case of θ-subsumption. This is an important property in the light of the upgrading

[4]In Progol, the θ-subsumption lattice is searched top-down but is bounded from below by a clause computed using inverse entailment.

procedure. For instance: the clause `class`(*positive*) :- `color-right`(*black*) is more general than `class`(*positive*) :- `color-left`(*white*), `color-right`(*black*).

At this point it is important to realize that θ-subsumption generalizes the well-known *turning constants into variables* introduced by [47]. For example, p :- q(X,Y), q(Y,X) is more general than p :- q(a,a) under θ-subsumption, but would not be regarded a generalization using the turning constants into variables. A second point where θ-subsumption generalizes Michalski's framework is that it also works for structured terms. E.g. p :- q(f(a)) is a specialization of p :- q(X).

Some more theoretical properties of θ-subsumption include (for more details see [53, 52, 49, 67]):

- it induces a quasi-order (reflexive and transitive) on the space of first order rules
- if c_1 θ-subsumes c_2 then $c_1 \models c_2$, i.e. c_1 logically entails c_2.
- there exist clauses $c_1 \neq c_2$ that are equivalent under θ-subsumption, e.g. p :- q(X,Y) and p :- q(X,Y),q(X,Z).
- the quasi-order can be turned into a partial order (also anti-symmetric) by defining equivalence classes in the usual way. There is then also a unique (up to variable renaming) representative of each equivalence class, which is called the *reduced* clause. The reduced clause r of a clause c is defined as the smallest subset of literals in c such that r is equivalent under θ-subsumption with c. E.g. p :- q(X,Y) is the reduced clause of p :- q(X,Y),q(X,Z).
- at the level of equivalence classes, one obtains a complete lattice, i.e. any two equivalence classes have a unique least upper bound (also called the least general generalization, the lgg) and a unique greatest lower bound.

4.5 Adapting the search operators

Now that we have chosen a framework for generality, we still need to define operators for searching the corresponding rule space. Given the advice of step 4, we will limit the discussion here to operators under θ-subsumption only.

Let us consider three typical operators used by concept learners. A specialization, resp. a generalization, operator that operates on a single clause, and a generalization operator that computes the least general generalization of two clauses.

The typical propositional specialization operator will basically add a condition to a rule. Using clauses, a condition can be added in two manners: either by adding a literal or by applying a substitution to the given clause. E.g. the clauses p :- q(X,Y),r(X), p :- q(X,X) and p :- q(X,a). are specializations of the clause p :- q(X,Y).
This yields the so-called refinement operators (cf. [52, 67]). There are some additional complications when using refinement operators wrt propositional systems:

- when simply adding literals, one might stay within the same equivalence class, and there might be infinite chains of such refinements, e.g. when refining p :- q(X,Y) to p :- q(X,Y), q(X,Z) and then to p :- q(X,Y), q(X,Z), q(X,W) ...

- it could be that even some proper refinements of a clause do not affect the coverage of the examples, this is known as the *determinacy problem* [58]. E.g. refining `class`(*pos*) :- `atom`(X) to `class`(*pos*) :- `atom`(X), `bond`(X,Y). As any atom will have bonds to other atoms, merely adding `bond`(X,Y) will not modify the coverage of the clause. This may misguide the heuristics of the learning engine.

Both difficulties can be alleviated by using a declarative (language) bias that will be discussed in the next section.

The typical propositional generalization operator will delete (or relax) a condition in a rule. Under θ-subsumption there are two ways of relaxing a clause: either delete a literal, or apply an inverse substitution to the clause. The first case is the easy one: e.g. generalize p :- q(X,Y), r(Y) towards p :- r(Y). The second case is more complicated and generalizes the turning constants into variables rule. If the constant (or term) to be generalized occurs only once in the clause, there is no problem: it can merely be generalized into a variable not yet occurring in the clause. E.g. p :- q(a,b) into p :- q(X,b). However, if the constant or term to be generalized occurs multiple times, generalization can be quite complex. Indeed, consider p :- q(a,a). This can be generalized into p :- q(X,X) or p :- q(a,X),q(X,a). One problem is the existence of infinite chains. I.e. the clause positive :- q(a,a) has the following generalizations: positive :- q(X,X), positive :- q(X,Y), q(Y,X), ... The most specific generalization is the infinite rule positive :- q(X_1,X_2), q(X_2,X_3),...,q(X_i,X_{i+1}),...).
This problem and the existence of a *lgg* operator (cf. below) explains why plain generalization operators are less popular in ILP than refinement operators. The problems with generalization operators can again be reduced by an appropriate declarative bias mechanism.

A third popular operator is the (generalization) operator that computes the least general generalization (*lgg*) of two clauses.

A clause g is a *least general generalization* (*lgg*) of the clauses C_1 and C_2
if and only if
g θ-subsumes C_1 and g θ-subsumes C_2, and for every clause g' such that
g' θ-subsumes C_1 and g' θ-subsumes C_2, g' also θ-subsumes g.

Plotkin has given a procedure to compute the *lgg* of two clauses: The *lgg* of two identical terms is the term itself ($lgg(t,t) = t$). The *lgg* of the terms $f(s_1,...,s_n)$ and $f(t_1,...,t_n)$ is $f(lgg(s_1,t_1),...,lgg(s_n,t_n))$. The *lgg* of the terms $f(s_1,...,s_n)$ and $g(t_1,...,t_m)$ where $f \neq g$ is the variable v where v represents this pair of terms throughout. The *lgg* of two atoms/literals $p(s_1,...,s_n)$ and $p(t_1,...,t_n)$ is $p(lgg(s_1,t_1),...,lgg(s_n,t_n))$, the *lgg* being undefined when the sign or the predicate symbols are different. Finally, the *lgg* of two clauses C_1 and C_2 is then

$\bigvee_{l_1 \in set(C_1), l_2 \in set(C_2)} lgg(l_1, l_2)$.
For example, the *lgg* of `father`(*luc*,*soetkin*) :- `parent`(*luc*,*soetkin*),`male`(*luc*),`female`(*soetkin*) and `father`(*jef*,*wim*) :- `parent`(*jef*,*wim*),`male`(*jef*),`male`(*wim*) is `father`(X,Y) :- `parent`(X,Y), `male`(X),`male`(Z).
The *lgg* is used in specific to general inductive logic programming systems like Golem [50]. The problem with the lgg operator is that the complexity of the lgg (i.e. the number of literals) may grow exponentially with the number of examples in the worst case.

Step 5 *: Use an operator under θ-subsumption. Use that one that corresponds closely to the propositional operator.*

In the ICL system, we choose a specialization operator under θ-subsumption. Due to the problems sketched above, we will embed it within a declarative bias mechanism.

4.6 The need for bias

In the previous section, several problems with pure θ-subsumption operators were mentioned. These problems are mainly due to the combinatorics of the search-space, the fact that the space is infinite rather than finite (as in the propositional case) and the determinacy problem. To make the search tractable and efficient, it is thus necessary to constrain the search space in some way. In ILP, this is solved using syntactical or semantical declarative bias mechanisms. Various formalisms exist (see [51] for an overview), but the overall idea is to limit the number of clauses considered. The most straightforward methods merely employ some bounds on the number of variables, or literals in clauses and make the search-space finite. Other methods will specify syntactic limitations on the clauses considered (from which an operator can be derived). E.g. using a number of schemata to enforce that clauses satisfy certain patterns, e.g. the pattern P(X,Y) :- Q(X), R(X,Y), where P, Q and R are 'predicate' variables (see [41]). Other methods use a kind of grammar construction to explicitly declare the range of acceptable clauses [16]. A third class of techniques uses so-called mode-declarations to state how clauses can be refined, like in PROGOL [48], TILDE [9, 7] and WARMR [26].

Step 6*: Use a declarative bias mechanism to limit the search-space.*

In ICL we selected the $\mathcal{D}$LAB declarative bias formalism of [20], which encodes a kind of grammar[5]. An example is given in Table 5. Min-Max:List means that at least Min and at most Max literals of List are allowed (`len` is the length of `List`). Note that shape(Object, 1-1:[circle,triangle]) is a shorthand for 1-1:[shape(Object,circle), shape(Object,triangle)]. Recursion is allowed. There is also a notion of dlab_variable (not used in the example) that allows the user

[5]We could have used as well the mode-declarations as in Tilde and Warmr.

Table 5: A $\mathcal{D}$LAB bias for the Bongard problem

```
dlab_template('
  1-len:[shape(Object1, 1-1:[circle,triangle]),
         size(Object1, 1-1:[small,large]),
         shape(Object2, 1-1:[circle,triangle]),
         size(Object2, 1-1:[small,large]),
         1-1:[in(Object1, Object2), left_of(Object1,Object2)]
        ]').
```

to define shortcuts for frequently occurring parts (like 1-1:[circle,triangle]).

Given a $\mathcal{D}$LAB expression, a refinement operator can be used to traverse the (restricted) search space. A complete refinement operator for $\mathcal{D}$LAB is given in [20]. For example, based on the $\mathcal{D}$LAB expression in Table 5, the top rule of the search space is
`class`($\oplus$) :- *true*.
The refinement operator will generate the following refinements for this rule:
`class`($\oplus$) :- `shape`(Object1, *circle*).
`class`($\oplus$) :- `shape`(Object1, *triangle*).
...
`class`($\oplus$) :- `left_of`(Object1, Object2).
These rules again can be refined further on. For the first rule:
`class`($\oplus$) :- `shape`(Object1, *circle*), `size`(Object1, *small*).
...
`class`($\oplus$) :- `shape`(Object1, *circle*), `shape`(Object2, *circle*).
...
Note that `class`($\oplus$) :- `shape`(Object1, *circle*), `shape`(Object1, *triangle*). is not a valid refinement.

The advantage of $\mathcal{D}$LAB is its expressive power. It allows the user to strongly bias the learning system ICL. On the other hand, a $\mathcal{D}$LAB expression can become very complex. Writing a $\mathcal{D}$LAB (and bias in general) is an iterative process and not always straightforward.
Note that some kind of lookahead can be performed by $\mathcal{D}$LAB to overcome the determinacy problem. Indeed, when using len-len:[List] in the template, all the literals in List must be added in one step as a refinement.

4.7 Implementing the algorithm

By now, we are ready to implement our first order learner. All basic modifications needed have been sketched. In this step, it is important that as many features of the original algorithm as possible are preserved, like search strategy, heuristics, noise-handling, pruning, parameters,... For example, ICL uses the same heuristics (Laplace estimate) as its propositional counterpart CN2.

Some advanced and specific features of propositional learning algorithms may need further changes. For example, discretization on numerical attributes cannot be mapped directly towards our relational representation (see Section 4.9).

Step 7: *Implement.*

Currently, ICL is implemented in MasterProLog (formerly ProLog by BIM) and freely available for academic use (runtime version for Solaris 2.5 is available on request).

4.8 Evaluation of ILP system

A first evaluation is testing the implementation on a propositional problem, for example the problem in Figure 1. The results should be compatible with the results of the corresponding propositional learner. In the ideal case these should be the same, but in reality some minor differences might occur. Many reasons exist: small differences in implementation, lack of some features (like handling of unknown values), a slightly different hypothesis space for the propositional and relational system,...

Next is learning on relational data that one is unable to learn on with a propositional learner. A good starting point is some artificial problem, like the Bongard problem in Figure 2, where one has already a solution (obtained by hand or by some other relational learner). Experiments with these data can give a good insight in the system (the parameters, the output, the behaviour of learning, some problems,...).

Then one can start the *real* work and run the system on real-like problems. A well-known application area is in molecular biology.

Step 8: *Evaluate your (first order) implementation on propositional and relational data.*

In Section 5, some experimental results with ICL will be discussed.

4.9 Extensions to the basic system

Many propositional systems have been extended with extra features or optimizations. These can also be incorporated in its relational counterpart, if possible. Some extensions can be plugged in as they are, others will need some adaption/upgrade similar to the other steps in the sketched methodology. In this way, ILP learners can reuse results from research on propositional learners.

Note that also ideas from other ILP systems can be incorporated. As such, results from ILP can be reused by propositional learners, so both communities can learn from each other.

Step 9: *Add interesting extra features.*

The system ICL has many extensions/optimizations w.r.t. the basic system described up to now (see [69, 68] for more details). To handle numerical data, we upgraded the discretization method of Fayyad and Irani (see [36, 29]) towards ICL. To handle multiple classes, we extended the CN2 method with a Bayes approach (inspired by [54]). This result can be integrated in CN2 without any problem (illustrating that results in the context of relational learners can be mapped back to propositional learners). Other extensions/optimizations of ICL include: learning both DNF and CNF theories, using the m-estimate (as in [32]) instead of the Laplace estimate as heuristic, extra pruning of the search space,... While CN2 has a specific handling of unknown values (* and ?), ICL just assumes the closed world assumption.

5 Some experimental results with ICL

To illustrate the utility of the method and the effectiveness of ICL, we will give an overview of some experiments performed with ICL.

5.1 Experimental settings

The experiments have been performed with ICL version 4.2, implemented in MasterProLog (formerly ProLog by BIM). We used a Sun Ultra 2 with two 167 Mhz UltraSPARC processors running Solaris 2.6, and a SUN Ultra 10 with a 333 Mhz UltraSPARCII processor running Solaris 7.

Unless stated otherwise, we used the default settings of ICL. The most important ones: significance level is 90%, heuristic is m_estimate (with parameter m the number of classes), the size of the beam is 5, and classes are *pos* and *neg*.

5.2 Propositional data

One of the nice properties of our methodology is its *backward compatibility*, meaning that the upgraded relational system behaves similar as its propositional predecessor on propositional data. However, some deflections occur due to differences in implementation.

To simulate CN2 with ICL, we can use the following simple $\mathcal{D}$LAB expression: 1-len:[att_1 = 1-1:[$v_{1,1}$, ..., v_{1,i_1}],..., att_k = 1-1:[$v_{k,1}$, ..., v_{k,i_k}]], with $v_{i,j}$ the values of the attribute att_i.

We have run ICL on a few propositional data sets used in [13]: voting-records, breast-cancer, lymphography and primary-tumor. Some information on the data sets is given in Table 6. We have chosen these data sets because they have no (or few) numerical values and only few unknown values. So this allows a close comparison.

We performed a similar experimental procedure as in [13]. The accuracies have been estimated by averaging the results over 20 runs (for each run, 2/3

Table 6: Details of the propositional domains used in the experiments. We did the same data conversions as documented in [13].

Domain	Number of			Unknown	Numerical
	Exs	Atts	Classes	values	values
voting-records	435	16	2	yes	no
breast-cancer	286	9	2	few	few
lymphography	148	18	4	no	few
primary-tumor	330	17	15	yes	no

Table 7: Comparison of ICL and CN2 on accuracy (with standard deviation) and rule set size (number of attribute tests/literals). Results for CN2 are taken from [13], Appendix 1.

Accuracy	ICL		CN2 (unordered)		ICL
Sign. Threshold	0%	99.5%	0%	99.5%	default settings
voting-records	94.1±1.5	92.5±2.0	94.8±1.8	93.3±2.1	94.1±1.9
breast-cancer	69.7±3.3	71.8±3.7	73.0±4.5	70.8±3.5	69.4±4.1
lymphography	80.3±4.0	76.2±6.3	81.7±4.3	76.5±5.3	81.9±5.8
primary-tumor	41.7±4.9	42.0±4.8	45.8±3.6	41.4±5.8	41.4±5.5

Rule set size	ICL		CN2 (unordered)		ICL
Sign. Threshold	0%	99.5%	0%	99.5%	default settings
voting-records	43.5	14.9	64.8	19.9	49.7
breast-cancer	158.5	17.4	100.5	18.0	136.1
lymphography	38.5	14.9	40.4	13.5	45.5
primary-tumor	267	115.35	351.0	131.4	253.6

of the data is selected randomly for training and the remainder for testing). The results are shown in Table 7. We have run ICL with the same language bias and the same settings as in [13]: beam=20 and heuristic=laplace. The last column gives the result of the default ICL performance (with default parameters: beam=5, significance level=90% and heuristic=m_estimate).

When we look at the accuracy and rule set size, we can conclude that ICL's performance is similar to CN2's, what we expected. Differences can be accounted by the small differences between CN2 and ICL w.r.t. options and implementation details There are however 2 boundary cases: for breast-cancer and primary-tumor with significance threshold 0% ICL performs less than CN2 w.r.t. accuracy. In both cases, the theories sizes also differ significantly. We haven't found any explanation for this[6].

[6]In other papers we have found similar results for CN2 as ours. For example in [28], the result for breast-cancer is 70.0±1.4 accuracy with a theory size of 114.5, and for primary-

Table 8: Accuracies for the four different backgrounds of the mutagenesis data (estimated by a 10-fold cross-validation). The first three columns are results for ICL (with the *default settings*, except for maxbody=8, and without discretization). Pos and Neg are the two classes and for each of them a DNF theory is learned and evaluated. Multi merges the 2 theories into a multi-class theory. The results for PROGOL, FOIL and TILDE have been taken from [8].

	Accuracies (%)					
	Neg	Pos	Multi	PROGOL	FOIL	TILDE
BG1	79.3±8.2	67.6±5.1	80.9±7.4	76	61	75
BG2	80.3±8.9	74.5±6.9	82.4±7.4	81	61	79
BG3	85.1±8.7	83.5±5.9	86.7±10.0	83	83	85
BG4	85.1±7.7	86.2±7.6	88.3±8.0	88	82	86

5.3 Relational data

ICL has been used in many experiments with (real-life) relational data sets. We will give some results, and refer to the literature for more details.

One of the most used data set in ILP is the **mutagenesis** one (see [64]). The data consists of 188 molecules, of which 125 are active (thus mutagenic) and 63 are inactive. A molecule is described by its atoms `atom(AtomID, Element, Type, Charge)` (the number of atoms differs between molecules, ranging from 15 to 35) and the bonds `bond(Atom1, Atom2, BondType)` between these atoms. Four different sets of background have been used (same as in [64]). Background 1 uses only the information on atoms and bonds, background 2 allows tests on the *charge* of an atom, background 3 adds 2 specific measures w.r.t. the molecule (e.g. $\log P$ and ϵ_{LUMO}) and background 4 consists of descriptions of higher-level structures that appear in the molecule (like aromatic rings).
Experiments with ICL on this data set can be found in [68]. Results with ICL version 4.2 are given in Table 8. We manually discretized the numerical values (i.e. $\log P$, ϵ_{LUMO} and the $Charge$ of the atoms). It seems that the multi-class theory is always better than the seperate (DNF) theory for each class. This is not so surprising as the multi-class theory combines the two seperate theories for each class, and resolves clashes between the two. The (multi-class) accuracy of ICL is significantly better than FOIL for background 1 and 2, and marginally better for background 3 and 4. ICL is also marginally better than PROGOL and TILDE for background 1. For Background 2, 3 and 4 however, the performance of ICL, PROGOL and TILDE are similar. Note that the accuracy increases as more background is added.

Results on the **biodegradability** domain can be found in [68] (preliminary results) and [34] (more recent results). The task is to predict the half-life in water for aerobic aqueous biodegradation of a compound from its chemical structure.

tumor the accuracy is 39.9±1.0 with a theory size of 302.8. These are similar to our results. The experiments in that paper used a beam of size 5 instead of 20. The other settings are the same.

Table 9: Accuracies of machine learning systems predicting Biodegradability. Results are taken from [34]. We have left out the results of the regression systems.

System	Representation	Accuracy	Accuracy (+/-)
C4.5	P1	55.2	86.2
C4.5	P2	56.9	82.4
RIPPER	P1	52.6	89.8
RIPPER	P2	57.6	93.9
FFoil	R1'	53.0	88.7
ICL	R1	55.7	92.6
SRT-C	P1 + R1	55.0	90.0
Tilde-C	R1	51.0	88.6
Tilde-C	P1 + R1	52.0	89.0

The biodegradation time has been discretized into 4 classes: fast, moderate, slow and resistant. The structure of a compound is represented by facts about atoms and bonds, much like in the mutagenesis domain. In [34] experiments on the relational data (denoted $R1$) and 2 propositional versions of the data (denoted $P1$ and $P2$) has been performed with the propositional classification systems C4.5 and RIPPER [15], and the relational learners FFoil [59], SRT [45], ICL and Tilde. A short overview of the results can be found in Table 9. Accuracy is classification accuracy and Accuracy (+/-1) is the accuracy where only misclassification by more than one class counts as an error (e.g. slow as fast, moderate as resistant,...). ICL has only been applied to the relational representation R1. Of all the relational learners using R1, ICL achieves the highest Accuracy and Accuracy(+/-1). Compared to the propositional systems, ICL is better than all systems in term of Accuracy(+/-1), except for RIPPER on P2. For more specific results and discussions we refer to the paper.

ICL also participated in the **PTE-2 challenge** of which the results have been published in [62, 63]. The challenge was to make carcinogenesis predictions for 30 compounds, based on models constructed by Machine Learning programs. There were 9 (legal) submissions using ILP systems (Tilde, Warmr/Maccent, ICL, P-Progol) and combinations of propositional systems (like C4.5) and ILP (like rules from Warmr). ICL and the other ILP systems perform unexpectedly well on scales of quantitative performance. ICL itself is in the top 3 of ILP systems (with 78% accuracy). ILP assisted models appear to be better than expert assisted ones (w.r.t. the PTE-2 data). Interesting results are obtained with propositional prediction methods using results from ILP systems (for example C4.5 using rules/sub-structures generated by Warmr).

Other successful experiments with ICL include: *finite element mesh design* by [68]; automated acquisition of knowledge on *traffic problem detection* by [30, 33]; and the problem of *diterpene structure elucidation* from ^{13}C NMR spectra by [30].

To conclude, ICL performs as well as other well-known ILP systems, and thus can be said to be a successful upgrade.

6 Related Work and Conclusions

There are plenty of other inductive logic programming systems whose development more or less fits in with the proposed methodology: FOIL [57], RIBL [35], SRT [45], TILDE [9, 7], WARMR [26, 25], MACCENT [24], jk-CT learner [21], CLAUDIEN [20], Probabilistic Relational Models [44], Cohen's Flipper (in [17]), [60] and RDBC [43].

E.g. Quinlan's FOIL can also be considered an upgrade of either Michalski's AQ (1983) or CN2, RIBL upgrades the classical k-nearest neighbor algorithm (using a first order distance due to [6]), SRT and TILDE upgrade the well-known decision (and regression) tree paradigm incorporated in CART [12] and C4.5 [55, 56], WARMR upgrades APRIORI [2, 1], MACCENT upgrades the Maximum Entropy approach in [5], De Raedt and Dzeroski's PAC-learning results (as well as its incorporation in the CLAUDIEN system) for jk-CT are derived from results in [66] for k-CNF, Reddy and Tadepalli's results are based on the well-known results on learning horn-sentences by [3], Flipper upgrades Cohen's earlier Ripper [15], Koller's probabilistic relational models upgrade (propositional) bayesian networks, and Kirste and Wrobel's cluster system upgrades bottom-up agglomerative clustering algorithms to first order logic.

Hence, it is clear that the methodology we presented is not really new. It has been applied - implicitly - several times before to obtain effective inductive logic programming systems. One might even argue that it has been applied in some of the pre-ILP work on relational learning (e.g. [70, 47]). The most important contribution of our work therefore is to describe the underlying recipe *explicitly* and to show through a case study that it can be used to obtain novel inductive logic programming techniques. By no mean we wish to imply that this recipe is the *only* way to obtain inductive logic programming systems. Certainly, some systems, of which perhaps PROGOL [48] and MIS [61] are the best examples of well-known inductive logic programming sytems that have been derived from logical principles (without our recipe). Yet, we hope that our work gives new insights into the field of inductive logic programming and its relation to propositional machine learning.

Acknowledgements

Wim Van Laer and Luc De Raedt are supported by the ALADIN Esprit Project 28.623.

The authors would like to thank the machine learning group of Leuven, and more specifically Hendrik Blockeel, Luc Dehaspe and Maurice Bruynooghe, for their feedback, comments and interesting discussions on this topic.

The breast cancer, lymphography and primary-tumor domain were obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data.

The Mutagenesis dataset is made public by King and Srinivasan [64] and is available at the ILP data repository [40].

More info on ICL can be found online:
`http://www.cs.kuleuven.ac.be/~wimv/ICL/main.html`.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. The MIT Press, 1996.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 207–216. ACM, Washington, D.C., USA, May 1993.

[3] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–162, 1992.

[4] F. Bergadano and D. Gunetti. An interactive system to learn functional logic programs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1044–1049. Morgan Kaufmann, 1993.

[5] A. Berger, V. Della Pietra, and S. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[6] G. Bisson. Conceptual clustering in a first order logic representation. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 458–462. John Wiley & Sons, 1992.

[7] H. Blockeel. *Top-down induction of first order logical decision trees.* PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1998.
`http://www.cs.kuleuven.ac.be/~ml/PS/blockeel98:phd.ps.gz`.

[8] H. Blockeel and L. De Raedt. Experiments with top-down induction of logical decision trees. Technical Report CW 247, Dept. of Computer Science, K.U.Leuven, January 1997. Also in Periodic Progress Report ESPRIT Project ILP2, January 1997. `http://www.cs.kuleuven.ac.be/-publicaties/rapporten/CW1997.html`.

[9] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.

[10] M. Bongard. *Pattern Recognition.* Spartan Books, 1970.

[11] I. Bratko and S. Muggleton. Applications of inductive logic programming. *Communications of the ACM*, 38(11):65–70, 1995.

[12] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[13] P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proceedings of the Fifth European Working Session on Learning*, pages 151–163. Springer-Verlag, 1991.

[14] P. Clark and T. Niblett. The CN2 algorithm. *Machine Learning*, 3(4):261–284, 1989.

[15] W. W. Cohen. Fast effective rule induction. In *Proceedings of the twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[16] W. Cohen. Grammatically biased learning: learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 68:303–366, 1994.

[17] L. De Raedt, editor. *Advances in Inductive Logic Programming.* IOS Press, 1996.

[18] L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95:187–201, 1997.

[19] L. De Raedt. Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 1–8. Springer-Verlag, 1998.

[20] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[21] L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

[22] L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1037–1042. Morgan Kaufmann, 1993.

[23] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the Sixth International Workshop on Algorithmic Learning Theory*, pages 80–94. Springer-Verlag, 1995.

[24] L. Dehaspe. Maximum entropy modeling with clausal constraints. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 109–124. Springer-Verlag, 1997.

[25] L. Dehaspe. *Frequent Pattern Discovery in First-Order Logic.* PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1998. `http://www.cs.kuleuven.ac.be/~ldh/`.

[26] L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 125–132. Springer-Verlag, Berlin, 1997.

[27] B. Dolsak, I. Bratko, and A. Jezernik. Finite element mesh design: An engineering domain for ilp application. In *Proceedings of the Fourth International Workshop on Inductive Logic Programming.* Gesellschaft für Mathematik und Datenverarbeitung MBH, Sankt Augustin, Germany, 1994.

[28] P. Domingos. A process-oriented heuristic for model selection. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 127–135. Morgan Kaufmann, San Francisco, CA, 1998.

[29] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the Twelfth International Conference on Machine Learning.* Morgan Kaufmann, 1995.

[30] S. Džeroski, S. Schulze-Kremer, et al. Diterpene structure elucidation from 13C NMR spectra with inductive logic programming. *Applied Artificial Intelligence*, 12(5):363–384, July-August 1998.

[31] S. Džeroski and I. Bratko. Applications of inductive logic programming. In L. De Raedt, editor, *Advances in inductive logic programming*, pages 65–81. IOS Press, 1996.

[32] S. Džeroski, B. Cestnik, and I. Petrovski. Using the m-estimate in rule induction. *Journal of Computing and Information Technology*, 1(1):37 – 46, 1993.

[33] S. Džeroski, N. Jacobs, M. Molina, and C. Moure. ILP experiments in detecting traffic problems. In *Proceedings of the Tenth European Conference on Machine Learning*, pages 61–66. Springer-Verlag, August 1998.

[34] S. Džeroski, H. Blockeel, et al. Experiments in predicting biodegradability. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming.* Springer-Verlag, 1999.

[35] W. Emde and D. Wettschereck. Relational instance-based learning. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, 1996.

[36] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann, San Mateo, CA, 1993.

[37] P. Flach. Strongly typed inductive concept learning. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 185–194. Springer-Verlag, 1998.

[38] J. Ganascia and Y. Kodratoff. Improving the generalization step in learning. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, pages 215–241. Morgan Kaufmann, 1986.

[39] F. Hayes-Roth and J. McDermott. An interference matching technique for inducing abstractions. *Communications of the ACM*, 21:401–410, 1978.

[40] D. Kazakov, L. Popelinsky, and O. Stepankova. ILP datasets page [`http://www.gmd.de/ml-archive/datasets/ilp-res.html`], 1996.

[41] J.-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive logic programming*, pages 335–359. Academic Press, 1992.

[42] R. King, M. Sternberg, A. Srinivasan, and S. Muggleton. Relating chemical activity to structure: an examination of ILP successes. *New Generation Computing*, 13(3-4):411–434, 1995.

[43] M. Kirsten and S. Wrobel. Relational distance-based clustering. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 261–270. Springer-Verlag, 1998.

[44] D. Koller. Probabilistic relational models. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, pages 3–13. Springer-Verlag, 1999.

[45] S. Kramer. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 812–819. AAAI Press/MIT Press, Cambridge/Menlo Park, 1996.

[46] R. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The AQ15 inductive learning system: an overview and experiments. In *Proceedings of IMAL 1986*. Université de Paris-Sud, Orsay, 1986.

[47] R. Michalski. A theory and methodology of inductive learning. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: an artificial intelligence approach*. Morgan Kaufmann, 1983.

[48] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[49] S. Muggleton and L. De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

[50] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.

[51] C. Nédellec, H. Adé, F. Bergadano, and B. Tausend. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press, 1996.

[52] S.-H. Nienhuys-Cheng and R. Wolf. *Foundations of inductive logic programming.* Springer-Verlag, 1997.

[53] G. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, pages 153–163. Edinburgh University Press, 1970.

[54] U. Pompe and I. Kononenko. Probabilistic first-order classification. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming.* Springer-Verlag, 1997.

[55] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[56] J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[57] J. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[58] J. Quinlan. Determinate Literals in Inductive Logic Programming. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 442–446. Morgan Kaufmann, 1991.

[59] J. Quinlan. Learning first-order definitions of functions. *Journal of Artificial Intelligence Research*, 5:139–161, 1996.

[60] C. Reddy and P. Tadepalli. Learning first-order acyclic Horn programs from entailment. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 23–37. Springer, 1998.

[61] E. Shapiro. An algorithm that infers theories from facts. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 446–452. Morgan Kaufmann, 1981.

[62] A. Srinivasan, R. D. King, S. H. Muggleton, and M. Sternberg. The predictive toxicology evaluation challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence)*, pages 1–6. Morgan Kaufmann, 1997.

[63] A. Srinivasan, R. King, and D. Bristol. An assessment of ILP-assisted models for toxicology and the PTE-3 experiment. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, pages 291–302. Springer-Verlag, 1999.

[64] A. Srinivasan, S. Muggleton, M. Sternberg, and R. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1,2), 1996.

[65] H. Toivonen, M. Klemettinen, et al. Pruning and grouping discovered association rules. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, pages 47–52, Heraklion, Crete, Greece, 1995.

[66] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[67] P. R. J. van der Laag and S.-H. Nienhuys-Cheng. Completeness and properness of refinement operators in inductive logic programming. *Journal of Logic Programming*, 34(3):201–225, 1998.

[68] W. Van Laer, L. De Raedt, and S. Džeroski. On multi-class problems and discretization in inductive logic programming. In *Proceedings of the Tenth International Symposium on Methodologies for Intelligent Systems*, pages 277–286. Springer-Verlag, 1997.

[69] W. Van Laer, S. Džeroski, and L. De Raedt. Multi-class problems and discretization in ICL (extended abstract). In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming*, pages 53–60, 1996.

[70] S. Vere. Induction of concepts in the predicate calculus. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 282–287. Morgan Kaufmann, 1975.

[71] C. Vrain. Ogust: A system that learns using domain properties expressed as theorems. In Y. Kodratoff and R. Michalski, editors, *Machine Learning: an artificial intelligence approach*, pages 360–381. Morgan Kaufmann, 1990.

[72] P. Winston. Learning structural descriptions from examples. In P. Winston, editor, *Psychology of Computer Vision*. The MIT Press, 1975.