# Multiple Predicate Learning in Two Inductive Logic Programming Settings

**Article** · March 1996

DOI: 10.1093/jigpal/4.2.227 · Source: DBLP



**2 authors:**

Luc De Raedt
KU Leuven
**615** PUBLICATIONS **14,241** CITATIONS
SEE PROFILE

Nada Lavrac
Jožef Stefan Institute
**409** PUBLICATIONS **10,230** CITATIONS
SEE PROFILE

# Multiple predicate learning in two Inductive Logic Programming settings

Luc De Raedt[1] and Nada Lavrač[2]

(1) Department of Computer Science,
Katholieke Universiteit Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

(2) Jožef Stefan Institute, Jamova 39,
61111 Ljubljana, Slovenia

November 8, 1995

**Abstract**

Inductive logic programming (ILP) is a research area which has its roots in inductive machine learning and computational logic. The paper gives an introduction to this area based on a distinction between two different semantics used in inductive logic programming, and illustrates their application in knowledge discovery and programming. Whereas most research in inductive logic programming has focussed on learning single predicates from given datasets using the normal ILP semantics (e.g. the well known ILP systems GOLEM and FOIL), the paper investigates also the non-monotonic ILP semantics and the learning problems involving multiple predicates. The non-monotonic ILP setting avoids the order dependency problem of the normal setting when learning multiple predicates, extends the representation of the induced hypotheses to full clausal logic, and can be applied to different types of application.

## 1 Introduction

Inductive logic programming (ILP) [29, 8, 33, 23] can be considered as the intersection of inductive machine learning and computational logic. From inductive machine learning, ILP inherits its goal: to develop tools and techniques to induce hypotheses from observations (examples) or to synthesize new knowledge from experience. By using computational logic as the representational mechanism for hypotheses and observations, ILP can overcome the two main limitations of classical inductive learning techniques (such as the TDIDT-family [39]):

- the use of a limited knowledge representation formalism (essentially propositional logic), and
- the difficulties to use substantial domain knowledge in the learning process.

The first limitation is important because many problems of various domains of expertise can only be expressed in a first-order logic and not in a propositional logic; this implies that there are inductive learning tasks for which no propositional learner (and none of the classical empirical learners) can be effective (see e.g. [31] and consider program synthesis). The difficulty to employ domain knowledge is also crucial because one of the well-established findings of artificial intelligence (and machine learning) is that the use of domain knowledge is essential to achieve intelligent behaviour. First results in applying ILP (cf. e.g. [35, 21, 16]) show that the use of logic as a representation mechanism for inductive systems is not only justified from a theoretical point of view, but also from a practical one.

From computational logic, ILP inherits not only its representational formalism, but also its theoretical orientation as well as some well-established techniques. Indeed, in contrast to most other practical approaches to inductive learning, ILP is also interested in properties of inference rules, in convergence (e.g. soundness and completeness) of algorithms and in the computational complexity of procedures. Furthermore, because of the common representation framework, ILP is relevant to computational logic, deductive databases, knowledge base updating, algorithmic debugging, abduction, constraint logic programming, program synthesis and program analysis, and vice versa.

In this paper, we investigate the different faces of ILP. We first discuss two different semantics for ILP: a normal semantics for ILP introduced by Plotkin [38] and followed by Muggleton [28] and incorporated in many well-known systems [29, 34, 40, 8, 9, 24], and a non-monotonic semantics derived from Helft's work [19] and used in [33, 12]. It is shown that the normal semantics leads to some problems when learning multiple predicates, and that these problems can be avoided using the non-monotonic semantics. The latter also allows for the induction of full first-order clausal theories. The two semantics are relevant to all ILP techniques. However, the paper is focussed on the so-called refinement (general to specific) techniques used in empirical ILP systems. Other faces of ILP, referring to different dimensions as perceived by users, are also discussed. We sketch some applications of ILP to knowledge discovery in databases [37] and programming and discuss their relation to the two semantics.

The paper is organized as follows. In Section 2 we specify the problem of ILP and study the two different ILP semantics. Section 3 surveys some ILP techniques and discusses different dimensions of ILP systems. Section 4 investigates refinement approaches to ILP, mostly in the normal ILP setting. In Section 5, we study the problem of multiple predicate learning in the normal ILP setting, whereas Section 6 outlines an approach to multiple predicate learning

in the non-monotonic ILP setting. Finally, in Section 7, we conclude and touch upon related work.

# 2 Problem specification

Roughly speaking, ILP starts from an initial background theory $B$ and some evidence $E$ (examples). The aim is then to induce a hypothesis $H$ that together with $B$ *explains* some properties of $E$. In most cases the hypothesis $H$ has to satisfy certain restrictions, which we shall refer to as the *bias*. Bias includes prior expectations and assumptions, and can therefore be considered as the logically unjustified part of the background knowledge. Bias is needed to reduce the number of candidate hypotheses. It consists of the language bias $L$, determining the hypothesis space, and the search bias which restricts the search of the space of the possible hypotheses. On the other hand, an inappriopriate bias can prevent the learner from finding the intended hypotheses.

In this section, we recall some logic programming concepts and use them to formally define the notions of hypothesis, theory, and evidence. Furthermore, we discuss two alternative semantics for ILP [33]: the normal ILP semantics [28], and the non-monotonic semantics [19, 12]. The labels 'normal' and 'non-monotonic' are mainly due to historical reasons. 'Normal' reflects the fact that this setting is the usual setting employed in ILP; the label 'non-monotonic' was introduced by Nicolas Helft because his setting employs a kind of closed world assumption, hence the relation to non-monotonic reasoning.

## 2.1 Some logic programming concepts

**Definition 1** *A clause is a formula of the form* $A_1, ..., A_m \leftarrow B_1, ..., B_n$ *where the* $A_i$ *and* $B_i$ *are positive literals (atomic formulae).*

The above clause can be read as $A_1$ or ... or $A_m$ if $B_1$ and ... and $B_n$. All variables in clauses are universally quantified, although this is not explicitly written. Throughout the paper, we assume that all clauses are *range restricted*, which means that all variables occurring in the head of a clause also occur in its body. Extending the usual convention for *definite clauses* (where $m = 1$), we call the disjunction of atoms $A_i$ the *head* of clause $c$ and denote it by *head*$(c)$. The conjunction of atoms $B_j$, called the *body* of clause $c$, is denoted by *body*$(c)$. A *fact* $A \leftarrow$ (or simply $A$) is a definite clause with an empty body ($m = 1,\ n = 0$). A set of clauses $T$ is called a *theory*, a set of definite clauses is referred to as a *definite theory*.

A definite theory $T$ has a unique least Herbrand model $M(T)$ (cf. [25]) which can, for the purposes of this paper, be defined as follows:

**Definition 2** *The least Herbrand model* $M(T)$ *of a definite theory* $T$ *is a set of ground facts*

$$M(T) = \{A \mid A \in \textit{Herbrand base of } T \textit{ and } T \models A\}.$$

*where the Herbrand base of $T$ is the set of all ground atoms which can be formed using the predicate, functor and constant symbols occuring in $T$.*

Thus, roughly speaking, the least Herbrand model is the set of all facts that are logically entailed by the theory. From a practical point of view, logical entailment of fact $A$ by theory $T$ can be verified using a PROLOG interpreter with knowledge base $T$ and query $? - A$.

A clause $c = A_1, ..., A_m \leftarrow B_1, ..., B_n$ is true in a model $M(T)$ if and only if for all substitutions $\theta$ grounding $c$ it holds that $\{B_1, ..., B_n\}\theta \subseteq M(T) \rightarrow \{A_1, ..., A_m\}\theta \cap M(T) \neq \emptyset$. This means that a clause is true in a model whenever the truth of the body (in the model) implies the truth of the head (in the model). From a more practical point of view, the truth of a clause $c$ in a model $M(T)$ can be verified by asserting $M(T)$ (or $T$) in a PROLOG knowledge base and running the query $? - \textit{body}(c),\ \textit{not head}(c)$. If the query fails, the clause $c$ is true in the model; if the query succeeds, it is false. These definitions are illustrated in Example 1.

**Example 1** *Consider the following definite theory $T$:*

$T$ = { flies(X) ← bird(X); bird(tweety) ← }

The least Herbrand model of $T$ is $M(T)$ = {flies(tweety); bird(tweety)}.

A full clause in this domain could be, for instance, ← abnormal(X),bird(X),flies(X), stating that abnormal birds do not fly. This clause is true in the model, as there are no facts for abnormal in $M(T)$ (the query fails). If however we would add abnormal(tweety) to the model, the clause would become false in this larger model (as the corresponding query would succeed).

## 2.2 Definition of two inductive logic programming settings

An *example* is a ground fact together with its truthvalue in the intended interpretation: positive examples are true ground atoms (i.e. of the form $e \leftarrow$ meaning $e \leftarrow \textit{true}$) and negative examples are false ground atoms (i.e. of the form $\leftarrow e$ meaning $\textit{false} \leftarrow e$). The sets of positive and negative examples are denoted as $E^+$ and $E^-$, respectively, and $E = E^+ \cup E^-$. In ILP, the background theory $B$ and hypotheses $H$ are represented by sets of clauses. For simplicity, we mainly focus on definite clauses for representing hypotheses and theories. Nevertheless, parts of our discussion extend to the more general (normal) program clauses [25], which are sometimes used in ILP systems [24, 40], as well as to (general) clauses, as used in [12]. Language bias imposes certain syntactic restrictions on the form of clauses allowed in hypotheses; for example, one might consider only constrained clauses [24]; these are clauses for which all

variables occurring in the body also occur in the head. For our purposes, we consider the language bias $L$ as a set of clauses allowed in hypotheses. Other restrictions frequently employed include abstract languages [8], determinations [41], schemata [20], inductive logic programming languages [2, 1], antecedent description grammars [7] and $ij$-determination [34].

Inductive logic programming can now be defined as follows:

**Given:**

- a set of examples $E$
- a background theory $B$
- a language bias $L$ that defines the clauses allowed in hypotheses
- a notion of *explanation* (a *semantics*)

**Find** : a hypothesis $H \subset L$ which explains the examples $E$ with respect to the theory $B$.

Let us now define the normal and non-monotonic settings of ILP, which determine different notions of explanation (semantics). Current research in ILP mainly employs the normal setting [28]:

**Definition 3** *The normal setting of ILP restricts $H$ and $B$ to sets of definite clauses, and employs the normal semantics: $B \cup H \models E^+$ and $B \cup H \cup E^- \not\models \square$.*

The above requirements are usually called 'global' completeness and consistency. They implicitly assume the the notion of *intensional coverage* defined as follows.

**Definition 4** *Given background theory $B$, hypothesis $H$ and example set $E$, an example $e \in E$ is (intensionally) covered by $H$ iff $B \cup H \models e$. Hypothesis $H$ is (globally) complete iff $B \cup H \models E^+$, i.e. if $\forall (e \leftarrow) \in E^+ : B \cup H \models e$. Hypothesis $H$ is (globally) consistent iff $B \cup H \cup E^- \not\models \square$, i.e. if $\forall (\leftarrow e) \in E^- : B \cup H \not\models e$.*

Given the restriction to definite theories $T$ for which there exists a unique least Herbrand model $M(T)$ and to ground atoms as examples, this is equivalent to requiring that all clauses in $E$ are true in $M(B \cup H)$, see [33].

**Example 2** *As an illustration, let $B$ = {bird(tweety); bird(oliver)}, $E^- = \emptyset$, and $E^+$ = {flies(tweety)}. A valid hypothesis in the normal setting is*

$$c_1 = \mathsf{flies(X)} \leftarrow \mathsf{bird(X)}$$

*Clause $c_1$ may contribute to a solution because it is* consistent, *i.e. for any substitution $\theta$ for which head$(c_1)\theta$ is false, body$(c_1)\theta$ is also false. Notice that $c_1$ realizes an inductive leap because $B \cup \{c_1\}$ entails* flies(oliver).

**Definition 5** *The non-monotonic setting of ILP, restricts $B$ to a set of definite clauses, $H$ to a set of (general) clauses, $E$ to positive examples, and requires that all clauses $c$ in $H$ are true in the least Herbrand model $M(T)$ where $T = B \cup E$*[1].

To illustrate the non-monotonic ILP setting, reconsider the above example.

**Example 3** *In the non-monotonic setting, clause* $c_1$ = flies(X) ← bird(X) is not a solution because there is a substitution $\theta$ = {X/oliver} for which $body(c_1)\theta$ is true (i.e. $\subset M(T)$) and $head(c_1)\theta$ is false in $M(T)$ (i.e. $\not\subset M(T)$). It is also complete. However, the clause $c_2$ = bird(X) ← flies(X) is a solution because for all X for which flies(X) is true, bird(X) is also true. This shows that the non-monotonic setting does not hypothesize properties not holding on the example set. Therefore the non-monotonic semantics realizes induction by *deduction*. Indeed, the induction principle of the non-monotonic setting states that the hypothesis $H$, which is *deduced* from the set of *observed* examples $E$ and the theory $B$ (using a kind of closed-world assumption), holds for *all* possible sets of examples (with a Herbrand base disjoint to the original set of examples). This realizes generalization beyond the observed examples (Herbrand base), which also results in an inductive leap. As a consequence of the induction by deduction operation, properties derived in the non-monotonic setting are more certain than those derived in the normal one[2].

## 2.3 Comparison of the two ILP settings

The differences between the normal and the non-monotonic setting are related to the closed-world assumption. In most applications of normal ILP [21, 35], only the set of positive examples is specified and the set of negative examples is derived from this by applying the closed-world assumption (cwa). In our illustration, this results in $E^-$ = {← flies(oliver)}. Given this modified $E^-$, clause $c_1$ cannot contribute to a solution in the normal setting because for $\theta$ = {X/oliver}, $head(c_1)\theta$ is false while $body(c_1)\theta$ is true. If, on the other hand, we ignore the difference between the background theory and the examples by considering the problem where $B = \emptyset$, $E^+$ = {flies(tweety), bird(tweety), bird(oliver)}, and $E^-$ = {← flies(oliver)} (cwa), clause $c_2$ is also a solution in the normal setting. Intuitively, this shows that solutions to problems in the normal setting, where the closed-world assumption is applied, are also valid in the non-monotonic setting (see [33] for a proof). Remember from the database literature that applying the closed-world assumption is only justified when the *universe* defined by the examples together with the theory is completely described (cf. also Example 4).

[1] Sometimes, see [33, 19], one also requires completeness and minimality. Completeness would mean that a maximally general hypothesis $H$ is found; minimality means that the hypothesis does not contain redundant clauses.

[2] These characteristics of the non-monotonic setting are clearer from the PAC-formulation of the non-monotonic setting [13], where each example corresponds to a different minimal model.

For example, in a medical application, all patients in the database should be completely specified, which means that all their symptoms and diseases should be fully described. Notice that this is different from requiring that the *complete* universe is described (i.e. all possible patients).

Solutions in the normal setting with the closed-world assumption are also solutions in the non-monotonic setting. The opposite does not always hold and this reveals the other main difference between the two settings. In the normal setting, the induced hypothesis can always be used to replace the examples because theory and hypothesis entail the observed examples (and possibly other examples as well). On the other hand, in the non-monotonic setting the hypothesis consists of a set of properties holding for the example set. There are no requirements nor guarantees concerning prediction. For instance, in the non-monotonic setting, clause $c_2$ is a solution for $B = \{\mathsf{bird(tweety)}, \mathsf{bird(oliver)}\}$ and $E^+ = \{\mathsf{flies(tweety)}\}$. Nevertheless, it cannot be used to predict the example in $E^+$.

The normal and non-monotonic semantics of ILP are further illustrated by Example 4 in a programming context and Example 5 in the context of knowledge discovery.

**Example 4** *(Programming)*
*Let the training examples consist of* $E^+ = \{\mathsf{sort}([1],[1]); \mathsf{sort}([2,1,3],[1,2,3])\}$ *and* $E^- = \{\leftarrow \mathsf{sort}([2,1],[1]); \leftarrow \mathsf{sort}([3,1,2],[2,1,3])\}$. *Let* $B$ *contain correct definitions for the predicates* permutation/2, *which is true if the second argument is a permuted list of the first argument, and* sorted/1, *which is true if its list-argument is sorted in ascending order. Given the normal setting, a possible solution to the inductive logic programming task could consist of the following clause* $c_3$*:*

$$c_3 = \mathsf{sort(X,Y)} \leftarrow \mathsf{permutation(X,Y),sorted(Y)}$$

*In the non-monotonic setting, using* $E^+$ *only, a solution could contain the following clauses:*

sorted(Y) ← sort(X,Y)
permutation(X,Y) ← sort(X,Y)
sorted(X) ← sort(X,X)

Whereas the normal setting results in a program for sorting lists, the non-monotonic setting results in a partial specification for the involved predicates. Notice that clause $c_3$ does not hold in the non-monotonic setting because the definitions of permutation and sorted also hold for terms not occurring in $E$ (which means that the closed-world assumption is not justified, cf. above). On the other hand, if we generalize the notion of a positive example to a definite clause and replace the evidence $E$ by a correct definition of the sort predicate

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *main line* | $J\clubsuit$ | $4\clubsuit$ | $Q\heartsuit$ | $3\spadesuit$ | $Q\diamondsuit$ | $9\heartsuit$ | $Q\clubsuit$ | $7\heartsuit$ | $Q\diamondsuit$ | $9\diamondsuit$ | ... |
| *side lines* | $K\clubsuit$ | $5\spadesuit$ | | | | $4\spadesuit$ | | $10\diamondsuit$ | | | |
| | | $7\spadesuit$ | | | | | | | | | |

Figure 1: An Eleusis layout.

(using for instance the definition of quick-sort), clause $c_3$ holds. This illustrates that the non-monotonic setting is applicable to reverse engineering.

**Example 5** *(Knowledge discovery)*
*Suppose we are playing the card game Eleusis, in which there are two players, and the first player decides upon a secret rule that defines legal sequences of cards. The first player shows the opponent a legal sequence of cards, which the second player has to complete, in accordance with the rule. For each guess (i.e., addition of a card to the sequence) by the second player, the first player says whether or not it results in a legal sequence. Consider the following partial sequences shown in Figure 1, where the intended rule is "Play alternate face and numbered cards" (cf. [40, 15]). If a card is a legal successor it is placed to the right of the last card in the sequence, otherwise it is placed under the last card. The horizontal main line represents the sequence as developed so far, while the vertical side lines show incorrect plays. The layout in Figure 1 is adapted from Quinlan (1990).*

*In ILP, one could have a predicate* can_follow(Card1,Card2), *which is true when* Card2 *is a legal successor of* Card1[3]. *The background knowledge about cards would define whether an individual card is face, numbered, red, black, etc.*

*In the normal ILP setting, we have positive and negative examples for the* can_follow *predicate. E.g. the first elements in the sequence would translate to* can_follow(jack-clubs,4-clubs), can_follow(4-clubs,queen-hearts),... *as positive examples, and* ← can_follow(jack-clubs,king-clubs), ← can_follow(4-clubs,5-spades), ← can_follow(4-clubs,7-spades),... *are negative examples. From the examples and background knowledge, the following secret rule (a definition of the* can_follow *predicate) can be induced:*

can_follow(X,Y) ← face(X), numbered(Y)
can_follow(X,Y) ← face(Y), numbered(X)

In the non-monotonic setting, one could induce (using only the positive examples, i.e. the main sequence) the following set of clauses:

← can_follow(X,Y), face(X), face(Y)

[3]In a more complicated setting, one would need to consider a sequence, not only the last card in the sequence.

← can_follow(X,Y), numbered(X), numbered(Y)
← can_follow(X,X)
face(Y) ← can_follow(X,Y), numbered(X)
numbered(Y) ← can_follow(X,Y), face(X)
...
One can see that in the normal setting classification rules are generated, whereas in the non-monotonic setting properties of legal card sequences are derived.

# 3 ILP techniques

Whereas Sections 2.2 and 2.3 gave a sketch of the two faces of inductive logic programming in logical terms, this section sketches the different faces of ILP in terms of techniques they employ, and consequently, in terms of their performance as observed by the user.

## 3.1 Dimensions of ILP

Practical ILP systems can be classified in five main dimensions, from a user perspective:

- *Empirical versus incremental.* This dimension describes the way the examples $E$ are obtained. In empirical ILP, the evidence is given at the start and not changed afterwards, in incremental ILP, the user supplies the examples one by one, in a piecewise fashion.

- *Interactive versus non-interactive.* In interactive ILP, the learner is allowed to pose questions to the user about the intended interpretation. Usually these questions query for the intended interpretation of examples or clauses.

- *Predicate invention allowed or not.* Predicate invention denotes the process whereby entirely new predicates (neither present in $E$ nor $B$) are induced. Predicate invention results in extending the vocabulary of the learner and may therefore facilitate the learning task. Although the idea of inventing new predicates is very appealing and some promising first approaches [30, 32, 46, 10, 44, 45] have been developed, the task of inventing new predicates is not well understood yet.

- *Single versus multiple predicate learning.* In single predicate learning, the evidence contains examples for only one predicate and the aim is to induce a definition for this predicate; in multiple predicate learning, the aim is to learn a set of possibly interacting predicate definitions or properties that hold among various predicates.

| System | Emp/Inc | Int/No | Inv/No | Sin/Mul | Noise/No |
|---|---|---|---|---|---|
| MIS | Inc | Int | No | Mul | No |
| CLINT | Inc | Int | Inv | Mul | No |
| MOBAL | Inc | No | Inv | Mul | No |
| CIGOL | Inc | Int | Inv | Mul | No |
| FOIL | Emp | No | No | Sin | Noise |
| GOLEM | Emp | No | No | Sin | Noise |
| LINUS | Emp | No | No | Sin | Noise |
| MPL | Emp | No | No | Mul | No |
| CLAUDIEN | Emp | No | No | Mul | No |

Table 1: Dimensions of ILP.

- *Noise-handling enabled or not.* Some ILP systems can only deal with exact, noiseless data whereas others contain elaborate mechanisms for dealing with real-life imperfect data, including random errors (called noise), see [23].

In Table 1 we sketch some well-known ILP systems along these five dimensions. The ILP systems sketched are: systems that induce predicate definitions in the normal ILP setting MIS [42], CLINT [8], MOBAL [20], CIGOL [32], FOIL [40], GOLEM [34], LINUS [24, 23], MPL [14] and, on the other hand, CLAUDIEN [12] which induces properties in the non-monotonic ILP setting. In the table, the following abbreviations are used: Emp/Inc (empirical/incremental), Int/No (interactive/non-interactive), Inv/No (predicate invention/no predicate invention), Sin/Mul (single predicate learning/multiple predicate learning), Noise/No (noise handling/no noise handling).

From Table 1 it follows that most of the systems are either *incremental multiple* predicate learners (systems MIS, CLINT, MOBAL and CIGOL) or *empirical single* predicate learners[4] (systems FOIL, GOLEM, LINUS). This means that essentially none of these ILP systems is applicable to knowledge discovery in databases which typically requires an empirical setting (all data are given in the database) and involves multiple predicates (as the different predicates in the database should be related to each other). Two novel extensions of FOIL, i.e. systems MPL and CLAUDIEN that can be regarded as empirical multiple predicate learners or knowledge discovery systems, are discussed in Sections 5 and 6.

[4]FOIL and GOLEM should not be regarded as multiple predicate learners, cf. [14] and Section 5.

## 3.2 Structuring the hypothesis space

As is the case for most problems of artificial intelligence, ILP can be regarded as a search problem. Indeed, the space of possible solutions is determined by the syntactic (language) bias $L$. Furthermore, there is a decision criterion (e.g. $B \cup H \models E^+$ and $B \cup H \cup E^- \not\models \square$ in the normal ILP setting) to check whether a candidate hypothesis $H$ is a solution to a particular problem. Searching the whole space is clearly inefficient, therefore structuring the search space is necessary. Nearly all symbolic inductive learning techniques structure the search by means of the dual notions of generalization and specialization [27, 11]. For ILP, there are syntactical [38] and semantical (logical) definitions [36, 4] of these notions:

**Definition 6** *(Semantic generalization) A hypothesis $H_1$ is semantically more general than a hypothesis $H_2$ with respect to theory $B$ if and only if $B \cup H_1 \models H_2$.*

**Definition 7** *(Syntactic generalization or $\theta$-subsumption) A clause $c_1$ (a set of literals) is syntactically more general than a clause $c_2$ if and only if there exists a substitution $\theta$ such that $c_1\theta \subseteq c_2$. A hypothesis $H_1$ is syntactically more general than a hypothesis $H_2$ if and only if for each clause $c_2$ in $H_2$ there exists a clause $c_1$ in $H_1$ such that $c_1$ is syntactically more general than $c_2$.*

Plotkin has proved that the relation *is more general than* of the syntactic notion of generalization induces a lattice (up to variable renamings and equivalence classes) on the set of all clauses. Notice that when a clause $c_1$ (resp. hypothesis) is syntactically more general than a clause $c_2$ it is also semantically more general. Clause false (denoted by $\square$) is maximally general for both notions of generalization.

## 3.3 Searching the hypothesis space

Both *is more general than* relations are useful for induction because:

- when generalizing a hypothesis $H_1$ to $H_2$, all formulae $f$ entailed by the hypothesis $H_1$ and background theory $B$ will also be entailed by hypothesis $H_2$ and theory $B$, i.e. $(B \cup H_1 \models f) \rightarrow (B \cup H_2 \models f)$.
- when specializing a hypothesis $H_1$ to $H_2$, all formulae $f$ logically not entailed by hypothesis $H_1$ and background theory $B$ will not be entailed by hypothesis $H_2$ and theory $B$ either, i.e. $(B \cup H_1 \not\models f) \rightarrow (B \cup H_2 \not\models f)$.

The two properties can be used to prune large parts of the search space. The second property is used in conjunction with positive examples. If a clause (together with $B$) does not entail a positive example, all specializations of that clause can be pruned, as they cannot imply the example. The first property is used with negative examples.

Most inductive learners use one of the search strategies below (cf. [11]):

- *General to specific* learners start from the most general clauses and repeatedly specialize them until they are no longer inconsistent with negative examples; during the search they ensure that the clauses considered entail at least one positive example. When building a hypothesis, learners employ a *refinement operator* that computes a set of specializations of a clause.

- *Specific to general* learners start from the most specific clause that implies a given example; they then generalize the clause until it cannot further be generalized without entailing negative examples. Very often, generalization operators start from a clause and a positive example not entailed by the clause; they then compute the starting clause (the most specific clause implying the example) for the example and compute the least general generalization of the two clauses (cf. [38]).

Both techniques repeat their procedure on a reduced example set if the found clause by itself does not entail all positive examples. They use thus an iterative process to compute disjunctive hypotheses consisting of more than one clause (often called the "covering" approach [26]).

Hypotheses generated by the first approach are usually more general than those generated by the second approach. Therefore the first approach is less cautious and makes larger inductive leaps than the second. General to specific search is very well suited for empirical learning in the presence of noise because it can easily be guided by heuristics. Specific to general search strategies seem better suited for situations where fewer examples are available and for interactive and incremental processing. In the rest of this paper, we only discuss general to specific learning; for specific to general learning we refer to [29, 8, 33].

# 4 Refinement techniques

Central in general to specific approaches to ILP is the notion of a refinement operator (first introduced by [42]):

**Definition 8** *A refinement operator $\rho$ for a language bias $L$ is a mapping from $L$ to $2^L$ such that $\forall c \in L : \rho(c)$ is a set of specializations of $c$ under $\theta$-subsumption*[5].

We call the clauses $c' \in \rho(c)$ the *refinements* of $c$. In general, refinement operators depend on the language bias they are defined for. As an example, consider the refinement operator for clausal logic of the definition below.

[5] Sometimes $\rho(c)$ is defined as the set of proper *maximally* general specializations of $c$ under $\theta$-subsumption, cf. [42, 12]

**Definition 9** *Clause $c' \in \rho(c)$, i.e. clause $c$ is a refinement of clause $c$, iff*

$$c' = \begin{cases} head(c), A \leftarrow body(c) & | \ A \text{ is an atom} \quad (1) \\ head(c) \leftarrow body(c), A & | \ A \text{ is an atom} \quad (2) \\ c\theta & | \ \theta \text{ is a substitution} (3) \end{cases}$$

*Refinements of type (1) are called* head refinements *of $c$ and refinements of type (2) are called* body refinements.

Refinements obtained using the above operator may not always yield proper specialisations but may be equivalent to the original clause.[6]

The rest of the paper is concerned with refinement techniques as applied in the normal and non-monotonic ILP setting. This section deals with refinement techniques for single predicate learning, whereas Sections 5 and 6 propose refinement techniques which can be applied when learning multiple predicates.

## 4.1 Refinement techniques in the normal ILP setting

Using refinement operators, it is easy to define a simple general to specific search algorithm for finding hypotheses. This is done in Algorithm 1, which is basically a simplification of the FOIL algorithm [40]. Notice that FOIL can be used to learn definitions of more than one target predicate. It learns function-free clauses using a refinement operator for DATALOG, i.e. the formalism of definite clauses with as only function symbols constants.

In a repeat loop, FOIL generates different clauses until all positive examples in $E_p{}^+$ for the selected target predicate $p$ are (extensionally) covered by the hypothesis. Once a clause is added to the hypothesis, all positive examples (extensionally) covered by that clause are deleted from the set of positive examples (for the definition of extensional coverage, see Section 4.2). To find one clause, FOIL repeatedly applies a body refinement operator (a variant of that in Definition 9 combining refinements of the form (2) and (3)) until the clause is consistent with all negative examples for the predicate. Though the search for the predicate definition of $p$ starts with the maximally general clause $p(X_1, ..., X_n)$ it is sufficient to consider only body refinements of this clause provided that one employs explicit unification. For instance, if two literals in the head should be unified, the condition $X_i = X_j$ would appear in the body. FOIL is heuristically guided as it does not consider all refinements of a too general clause, but only the *best* one (the criterion is based on information theory, see [40]). This amounts to a *hill-climbing* search strategy. A variant of this algorithm employing beam-search is described in [17].

FOIL is together with GOLEM [34], which works specific to general instead of general to specific, one of the best known empirical ILP systems. Both systems are very efficient and have been applied to a number of real-life applications of *single predicate learning* [17, 3, 35, 21]. Notice that splFOIL$(B, H_p, E_p)$ in

[6] In theory one should work with reduced clauses only, cf. [38].

```
procedure FOIL(B,H,E)
hypothesis H := ∅
for all predicates p occurring in E do
        call splFOIL(B,H_p,E_p)
        H := H ∪ H_p
endfor


procedure splFOIL(B,H_p,E_p)
E_p^+ := the set of all positive examples for p
E_p^- := the set of all negative examples for p
hypothesis H_p := ∅
repeat
        clause c := p(X_1,...X_n) ←; where all X_i are different variables
        while c extensionally covers examples from E_p^- do
                build the set S of all body refinements of c
                c := the best element of S
        endwhile
        add c to H_p
        delete all examples from E_p^+ extensionally covered by c
until E_p^+ is empty
```

Algorithm 1: A simplified FOIL algorithm.

Algorithm 1 (spl stands for single predicate learning) learns a predicate definition for a single target predicate $p$ and that the procedure FOIL$(B, H, E)$ is the simplistic procedure employed by FOIL for learning multiple predicates (see the proposed improvements of this algorithm in Section 5). However, since they employ the normal ILP setting, they suffer from some problems. These problems are of two types:

- the extensionality problem, which is specific to GOLEM and FOIL, is due to the incorrect definition of coverage, which is in turn due to working with an extensional background model (see [14, 2, 5]),
- the order dependency problem when learning multiple predicates and/or recursive predicate definitions.

The first problem is specific to FOIL and GOLEM, whereas the second problem holds for the normal ILP setting in general; it is discussed in the context of multiple predicate learning in Section 5.

## 4.2 The extensionality problem

The problems with extensional models are well-known and have already been published in the ILP literature [2, 5, 14]. In summary, to test whether a hypothesis $H$ covers an example $e$, extensional methods use the following extensional coverage test.

**Definition 10** *A hypothesis $H$ extensionally covers an example $e$ iff there exists a clause $c \in H$ and a substitution $\theta$ such that $c\theta$ is ground, $head(c)\theta = e$ and $body(c)\theta \subseteq M(B)$, where $M(B)$ is the least Herbrand model of background knowledge $B$*[7].

Given this extensional coverage test, for example, the tautology consisting of $p(X) \leftarrow p(X)$[8] is always complete and consistent because the positive examples are always added to the model. For more details, and less trivial examples, we refer to [2, 5, 14].

Problems with extensionality are avoided by taking the normal intensional coverage test $B \cup \{c\} \models e$ for $e \in E$, instead of the extensional coverage test $\exists\theta$: $head(c)\theta = e$ and $body(c)\theta \subseteq M(B)$ used in FOIL and GOLEM. Notice that the extensional background knowledge in FOIL and GOLEM actually means that $B$ itself is a set of ground facts, i.e., $B = M(B)$. Moreover, even if no extensional background model were employed, FOIL and GOLEM would still yield incorrect results, since (in FOIL) the coverage test $B \cup \{c\} \models e$ would be 'local' to a

[7] In case that the model $M(B)$ is infinite, then a finite $h$-easy model is assumed, which contains all ground facts that can be derived from $B$ using a depth-bounded proof procedure of a selected depth $h$, see [34].

[8] This clause is usually excluded from the hypothesis space by systems such as FOIL and GOLEM.

clause $c$ and $B$, instead of testing the coverage 'globally' using $B \cup H \cup \{c\} \models e$ ('global' with respect to $B$ and the entire hypothesis $H \cup \{c\}$ built so far). One could try to improve upon FOIL, by changing the extensional tests for coverage into intensional coverage tests. Nevertheless, the resulting algorithm would still suffer from the problem of overgeneralization. This problem is dealt with extensively in Section 5.2.

# 5 Learning multiple predicates in the normal ILP setting

In the multiple predicate learning task, $E$ contains examples for $m$ target predicates $p_1, ..., p_m$. Therefore, $E$, $E^+$ and $E^-$ are divided into $m$ subsets $E_{p_i}$, $E^+_{p_i}$ and $E^-_{p_i}$ according to these predicates. The predicates whose definitions are given initially in the background theory $B$ are denoted as $q_1, ..., q_l$ ($p_i \neq q_j$).

By now, a hypothesis $H_p$ was defined as a set of definite clauses for a predicate $p$. In the multiple predicate learning task, the notion of an m-hypothesis is introduced.

**Definition 11** *An* m-hypothesis $H$ *is a set of definite clauses for $m$ different predicates $p_1, ..., p_m$. An m-hypothesis $H$ is the union of $m$ hypotheses $H_{p_i}$ for $i = 1, ..., m$, $H = \cup_{i=1}^{m} H_{p_i}$.*

The definition of intensional coverage, completeness and consistency can now be trivially adapted from Definition 4. An m-hypothesis $H$ intensionally covers an example $e$ given background theory $B$ iff $B \cup H \models e$. An m-hypothesis $H$ is (globally) complete iff $B \cup H \models E^+$, and (globally) consistent iff $B \cup H \cup E^- \not\models \square$.

Assuming the hypothesis language of definite clauses, multiple predicate learning in the normal ILP semantics is defined as follows:

**Given:**

- a set of training examples $E$ for $m$ different predicates $p_1, ..., p_m$, and
- a background theory $B$, containing predicate definitions for $q_1, ..., q_l$

**Find:** an $m$-hypothesis $H$ which is globally complete and consistent with respect to $E$ and $B$.

The multiple predicate learning task is denoted by a triple $mpl(B, H, E)$. It is illustrated in Example 6. A single predicate learning task is a special case of the above problem specification where $m = 1$. For a predicate $p_i$, this task is denoted by $spl(B, H_{p_i}, E_{p_i})$.

To elaborate our ideas in more detail, we need to make a further distinction between 'local' and 'global' properties. Global properties of hypotheses

and clauses are defined in the context of the entire example set $E$ and the $m$-hypothesis $H$ whereas local properties are defined using $E_{p_i}$ and $H_{p_i}$:

**Definition 12** *Hypothesis $H_{p_i}$ is locally complete iff $B \cup H_{p_i} \models E^+_{p_i}$. Hypothesis $H_{p_i}$ is locally consistent iff $B \cup H_{p_i} \cup E^-_{p_i} \not\models \square$.*

**Example 6** *(Multiple predicate learning)*
*We illustrate the problem of multiple predicate learning on a relatively simple task where the background theory $B$ is extensional and contains a complete set of facts for the background predicates* parent, female *and* male.

*The background theory $B$ contains:*

male(prudent) ←
male(willem) ←
male(etienne) ←
male(leon) ←
male(rene) ←
male(bart) ←
male(luc) ←
male(pieter) ←
male(stijn) ←

female(laura) ←
female(esther) ←
female(rose) ←
female(alice) ←
female(yvonne) ←
female(katleen) ←
female(lieve) ←
female(soetkin) ←
female(an) ←
female(lucy) ←

parent(bart,stijn) ←
parent(bart,pieter) ←
parent(luc,soetkin) ←
parent(willem,lieve) ←
parent(willem,katleen) ←
parent(rene,willem) ←
parent(rene,lucy) ←
parent(leon,rose) ←
parent(etienne,luc) ←
parent(etienne,an) ←
parent(prudent,esther) ←

parent(katleen,stijn) ←
parent(katleen,pieter) ←
parent(lieve,soetkin) ←
parent(esther,lieve) ←
parent(esther,katleen) ←
parent(yvonne,willem) ←
parent(yvonne,lucy) ←
parent(alice,rose) ←
parent(rose,luc) ←
parent(rose,an) ←
parent(laura,esther) ←

*Given the above background theory $B$, the multiple predicate learning task is to learn* father, mother *and* ancestor *from the complete set of examples for these predicates, under the usual interpretation of* father, mother *and* ancestor. *Here completeness means that all positive and negative examples of the predicates are available to the system (negative examples are generated under the cwa). A complete and consistent 3-hypothesis $H$ is:*

ancestor(X,Y) ← parent(X,Y)
ancestor(X,Y) ← parent(X,Z) ∧ ancestor(Z,Y)
father(X,Y) ← parent(X,Y) ∧ male(X)
mother(X,Y) ← parent(X,Y) ∧ female(X)

procedure **FOIL**($B$,$H$,$E$)
hypothesis $H := \emptyset$
**for** all predicates $p_i$ occurring in $E$ **do**
    call **splFOIL**($B \cup H$,$H_{p_i}$,$E_{p_i}$)
    $H := H \cup H_{p_i}$
**endfor**

Algorithm 2: A rudimentary modification of the top-level FOIL algorithm.

Notice that learning multiple predicates from examples is an important problem. One significant application is in knowledge discovery, where one faces different phenomena and needs to relate them to each other. Another interesting application is logic program synthesis.

## 5.1 The order dependency problem

The most straightforward way to solve a multiple predicate learning task $mpl(B, H, E)$ using a single predicate learner works as follows. Multiple predicate learning starts with an empty hypothesis, i.e. $H = \emptyset$. In each step of multiple predicate learning, the current hypothesis $H_{cur} = H_{p_1} \cup \ldots \cup H_{p_k}$, $k < m$, needs to be refined. The next predicate $p_{k+1}$ ($p_{k+1} \neq p_j$ for all $j$: $1 \leq j \leq k$) is selected and a new hypothesis $H_{p_{k+1}}$ is constructed by solving the single predicate learning task $spl(B \cup H_{cur}, H_{p_{k+1}}, E_{p_{k+1}})$, where $E_{p_{k+1}}$ is the training set for predicate $p_{k+1}$. The background knowledge to be considered in single predicate learning is not only $B$, but $B$ together with the hypothesis $H_{cur}$ learned so far. This idea can be implemented in FOIL in Algorithm 2:

Notice that in the actual implementation of FOIL, outlined in Section 4.1., the splFOIL is called with splFOIL($B$, $H_{p_i}$, $E_{p_i}$) instead of splFOIL($B \cup H$,$H_{p_i}$,$E_{p_i}$). We now show that this algorithm encounters a number of problems. The main problem in multiple predicate learning is to determine the predicate to be learned next. This is a serious problem because processing the predicates in a non-optimal order may significantly affect the learning results: no solution may be found or the induced hypothesis may be very complex.

The reason for the impact of the order of learning the predicates on the learning results is the use of a background theory. This theory determines the difficulty of the learning task. Given all relevant predicates in the background theory, the single predicate learning task is easy. On the other hand, if the available predicates are less relevant or irrelevant, the learning task becomes hard or unsolvable. For instance, learning the predicate grandfather in terms of male, female and grandparent is easy; using male, female and parent is harder, and using only male and female the problem is unsolvable.

The situation is even more complicated than indicated above. All ILP (and other inductive) learners employ an explicit or implicit bias, which effectively restricts the space of considered hypotheses. Given a language bias $L$ of definite or normal program clauses, even for a fixed number of predicates of arity 2 or larger, the number of different clauses is infinite. Since no system can search forever, additional bias is needed to make the search space finite and to consider the most interesting clauses first. As examples of such biases, consider GOLEM's $ij$-determination (explicit declarative bias) and FOIL's encoding length restriction (implicit procedural bias). Bias makes the search space manageable, but can prevent the learner from finding the desired predicate definitions. When trying to learn multiple predicates using a single predicate learner, bias and ordering effects are both to be taken into account. For certain orderings of learning the predicates, the learning task as a whole may be solvable, whereas for others, it may not. Clearly, in the worst case, which arises when there exists only one order of learning the predicates that leads to a solution, all possible orderings of learning the predicates need to be considered (formulated more precisely, an ordering is a sequence of the learning tasks $spl(B \cup H_{cur}, H_{p_i}, E_{p_i})$ for the current hypothesis $H_{cur}$ and there is precisely one task for each predicate $p_i$). This is clearly computationally very expensive. Even when the worst case does not arise, using a non-optimal order may have negative effects on the results, in the sense that the induced hypotheses may be unnecessarily complex and thereby also less reliable and accurate.

## 5.2 Overgeneralization

When an overly general clause $c$ is learned for a predicate $p_i$ and $p_i$ has to be used in a clause for a predicate $p_j$, overgeneralization may prevent the learner from finding globally consistent clauses for $p_j$. The reason is that the decision to add clause $c$ to the theory is made at a "local" level, based on the examples for $p_i$ only (see Definition 12). When using this predicate later, there are also global effects. One effect may be that $p_j$ falls off the learnability cliff [46] (all clauses for $p_j$ may be overly general, or may not satisfy the bias restrictions). As an example, consider GOLEM on the task of Example 6, with the order of learning father, mother and ancestor. Assume that – due to too few negative examples for father – the learned definition for father is overly general (e.g. father(X,Y) ← parent(X,Y)). In this case the definition of ancestor is not determinate again. The example implies that overgeneralization of one predicate can prevent the learner from finding the definitions of other predicates. It is hard to see how global effects can be taken into account by single predicate learners, which make local decisions.

## 5.3 Mutually recursive predicates

Mutually recursive predicates[9] complicate the learning process because the learning of one (mutually recursive) predicate should be interleaved with the learning of the other ones. To learn multiple mutually recursive predicates, one should – in the worst case – not only consider all orderings of the single predicate learning tasks, but all their sequences (i.e. single predicate learning tasks may have to be considered more than once).

Because of the problems outlined above, the FOIL algorithm (both Algorithms 1 and 2) does not work properly. A modified algorithm is given in Algorithm 3. Two modifications are incorporated. First, it tests intensional coverage of hypotheses at the global level instead of extensional coverage. And second, it implements a rudimentary approach of dealing with the order dependency problem by providing for a backtracking mechanism. Two forms of backtracking are needed. First, because local changes may have global effects, it is necessary to backtrack on $H$ whenever it becomes "globally" inconsistent. Second, for reasons outlined above, backtracking on the order of learning the predicates is provided as well. Notice however that backtracking may be computationally very expensive.

## 5.4 The MPL algorithm

To avoid the combinatorial explosion due to backtracking in Algorithm 3, we have developed heuristics which are incorporated in the MPL algorithm [14], sketched in Algorithm 4. Note that Algorithm 4 contains a simplified version of MPL in order to keep the exposition simple. For full details on MPL, we refer to [14].

The main differences between the mplFOIL approach and the MPL algorithm [14] are the following:

- MPL performs a hill-climbing strategy on hypotheses: in each step of this cycle it induces a clause; if adding the induced clause to the current hypothesis results in inconsistencies, MPL removes some clauses from the current hypothesis; otherwise it continues with the updated current hypothesis to learn clauses for non-covered positive examples, if any.

- To learn one clause, MPL employs a beam-search similar to mFOIL [17]. The only modification is that rather than storing (definite) clauses and computing their best refinements, MPL keeps track of bodies of clauses, and selects at each step the best body refinements. The quality of a body is defined here as the quality of the best clause in its head refinements. Computing the body first and selecting its head later allows to dynamically determine the next predicate to be learned. In this way, the order in

[9] Although mutually recursive predicates are not frequently needed, mutual recursion cannot always be avoided (without introducing additional new predicates).

```
procedure mplFOIL(B,H,E)
hypothesis H := ∅
for all predicates p occurring in E do
        E_p^+ := the set of all positive examples for p
        E_p^- := the set of all negative examples for p
        repeat
                clause c := p(X_1, ...X_n) ←; where all X_i are different variables
                while B ∪ H ∪ {c} ∪ E_p^- ⊨ □ do
                        build the set S of all body refinements of c
                        c := the best element of S
                endwhile
                add c to H
                if B ∪ H ∪ E^- ⊨ □
                then  backtrack on H or on the order of selecting the predicates
                endif
                delete all examples e ∈ E_p^+ such that B ∪ H ∪ {c} ⊨ e
        until E_p^+ is empty
endfor
```

Algorithm 3: An improved FOIL algorithm for dealing with recursion and multiple predicates.

```
procedure MPL(B,H,E)
hypothesis H := ∅
repeat
        body(c) := true
        c' := best head refinement of c
        while B ∪ H ∪ {c'} ∪ E⁻ ⊨ □ do
                build the set of all body refinements of c
                c' := best head refinement of c
        endwhile
        add c' to H
        delete all examples from E⁺ entailed by B ∪ H
        if B ∪ H ∪ E⁻ ⊨ □
        then    remove minimal set of clauses S from H such that B ∪ H ∪ E⁻ ⊭ □
                add back to E⁺ all (positive) examples previously covered by S and not by H
        endif
until E⁺ is empty or no improvement possible on H
```

Algorithm 4: A simplified MPL algorithm.

which predicates are learned is determined heuristically. A similar idea was implemented in the CN2 algorithm [6].

- When estimating the quality of a clause, MPL does not only take into account the examples of the predicate in the head of the clause, but also the other ones, which reduces the number of overgeneralizations. To be more precise, MPL employs a weighted average of the local and the global Laplace estimates [6, 23].

- When overgeneralisations arise, some clauses are deleted, and the learning process is restarted. Care is taken not to get into infinite loops by first deleting clauses and then adding the same ones again.

Preliminary experiments have shown that MPL correctly induces hypotheses for some multiple predicate learning problems beyond the scope of the current empirical ILP approaches, cf. [14]. The experiments have also shown that the MPL approach is computationally quite expensive. The reasons for this are twofold. First, MPL works intensionally resulting in a need for backtracking (which is still expensive though it is implemented in a more clever way than in Algorithm 4). Second, to learn one clause, it needs to construct all head refinements of a body. This means that instead of evaluating one clause, it evaluates $m$ clauses where $m$ is the number of target predicates. As a conclusion, we believe that our results for multiple predicate learning in the normal setting

for inductive logic programming are mostly negative as a correct (even heuristic) treatment of the problem blows up the search space.

# 6 Learning multiple predicates in the non-monotonic ILP setting

Many of the above sketched problems disappear when adopting the non-monotonic ILP setting. Indeed, the main problems in the normal setting are concerned with order dependency and dependencies among different clauses. Using the non-monotonic semantics these problems disappear because if $H_1$ is true in a model and $H_2$ is true in the same model, then their conjunction $H_1 \cup H_2$ is also true. Therefore clauses can be investigated completely independently of each other (they could even be searched in parallel): it does not matter whether $H_1$ is found first or $H_2$. Furthermore, interactions between different clauses are no longer important: if both clauses are individual solutions, their conjunction is also a solution. This property holds because the hypothesis is produced deductively in the non-monotonic setting. An important consequence is that the search of hypotheses reduces from $2^L$ to $L$. This is in contrast to the normal ILP setting, which allows to make inductive leaps, whereby the hypothesis together with the theory may entail facts not in the evidence. As different clauses in the normal setting entail different facts, the order of inducing and the way of combining different clauses in the normal setting affects the set of entailed facts.

A key observation underlying the non-monotonic ILP system CLAUDIEN [12] is that clauses $c$ that are false on the minimal model of $B \cup E$ are overly general, i.e. that there exist substitutions $\theta$ for which $body(c)\theta$ is true and $head(c)\theta$ is false in the model. In Section 5 we saw how overly general clauses could be specialized by applying refinement operators. The same applies here. In particular, body refinements decrease the number of substitutions for which $body(c)$ holds whereas head refinements increase the number of substitutions for which $head(c)$ holds.

Based on this, the CLAUDIEN algorithm (see Algorithm 5) was designed to induce clausal theories from databases. CLAUDIEN starts with a set of clauses $Q$, initially only containing the most general clause false and repeatedly refines overly general clauses in $Q$ until they satisfy the database. CLAUDIEN prunes away clauses already entailed by the found hypothesis at point (1) because neither they nor their refinements can result in new information[10]. Furthermore, at point (2) it tests whether refinements are relevant. Roughly speaking, relevance means that at least one substitution is handled differently by the refinement $c'$ than by its parent clause $c$. Under certain conditions on the language bias and refinement operator discussed in [12], irrelevant clauses can safely be pruned away. Intuitively, an irrelevant refinement $c'$ of $c$ is a refinement for which $c$ and

[10] This is verified using Stickel's fast theorem prover [43].

$Q := \{false\}$; $H := \emptyset$;
**while** $Q \neq \emptyset$ **do**
    delete $c$ from $Q$
    **if** $c$ is true in the least model of $B \cup E$
    **then** **if** $H \not\models c$ (1)
        **then** add $c$ to $H$
        **endif**
    **else** **for** all relevant $c' \in \rho(c)$ (2) **do**
            add $c'$ to $Q$
        **endfor**
    **endif**
**endwhile**

Algorithm 5: A simplified CLAUDIEN.

$c'$ are logically equivalent on the least model of $B \cup E$. Notice also that CLAUDIEN does not employ a heuristic search strategy, but a complete search of the relevant parts of the search space. Complete search[11] allows to find all properties expressible in the given language bias, thereby the most general hypothesis explaining the database is found.

# 7 Experimental results

This section reports on a number of results produced using actual ILP systems such as FOIL 2.1[12], MPL and CLAUDIEN. Though the experiments are very simple and use small toy problems, they clearly show that the theory of multiple predicate learning presented in this paper is in line with the practice of inductive logic programming. As such, the problems indicated above may arise in practice and the solutions outlined do - indeed - solve some of the problems.

## 7.1 Experiments with FOIL

In this section we show that using extensional coverage in FOIL can lead to problems because an example may be covered intensionally, but not extensionally and vice versa. Experiments were performed using FOIL2.1.

[11] In the implementation, we employ depth-first iterative deepening [22].

[12] As these experimented were carried out some time ago, an early version of FOIL was used. We did not verify that more recent version of FOIL would produce the same results, as the experiments are merely meant to show that the theory of multiple predicate learning presented above is in line with its practice.

**Experiment A: Extensional consistency, intensional inconsistency**

Reconsider the background theory specified in Example 6. Assume the example set is specified as follows: $E$ contains all positive examples for father, and as negative examples for father all facts of the form father(a,b) for which parent(a,b) is false; for the predicate male_ancestor $E$ contains all possible positive and negative examples. In this case, the following $m$-hypothesis is extensionally globally complete and consistent, but not intensionally consistent:

father(X,Y) ← parent(X,Y)
male_ancestor(X,Y) ← father(X,Y)
male_ancestor(X,Y) ← male_ancestor(X,Z), parent(Z,Y)

Indeed, male_ancestor(lieve,soetkin) is covered intensionally but not extensionally. The result of this is that the $m$-hypothesis is intensionally globally inconsistent.

**Experiment B: Extensional completeness, intensional incompleteness**

Reconsider the theory of Example 6 and suppose the aim is to induce the concepts male_ancestor and female_ancestor from father and mother and a complete example set. The definition below, induced by FOIL2.1, is complete and consistent from an extensional point of view (in the given database, both parents are always specified). From an intensional point of view, however, the definition is useless as it cannot be used to derive conclusions from a theory containing father and mother only.

male_ancestor(X,Y) ← father(X,W), mother(M,W), female_ancestor(M,Y)
female_ancestor(X,Y) ← mother(X,W), father(F,W), male_ancestor(F,Y)

**Experiment C: Intensional completeness, extensional incompleteness**

Reconsider the background theory specified in Example 6. Assume the following example set:

$E^+$ = { father(luc,soetkin), father(bart,stijn), male_ancestor(rene,lieve),
male_ancestor(bart,stijn), male_ancestor(luc,soetkin) }

$E^-$ = { ← father(lieve,soetkin), ← male_ancestor(esther,lieve),
← male_ancestor(katleen,stijn) }

In this case, the following $m$-hypothesis is intensionally globally complete and consistent, but not extensionally complete:

father(X,Y) ← male(X), parent(X,Y)
male_ancestor(X,Y) ← father(X,Y)
male_ancestor(X,Y) ← male_ancestor(X,Z), parent(Z,Y)

Now, male_ancestor(rene,lieve) is covered intensionally but not extensionally.

## 7.2 Experiments with MPL

Using a prototype implementation of the MPL algorithm in PROLOG, two experiments were performed on the domain of learning family relations from Example 6. The details of this experimental setting can be found in [12].

### Experiment 1

The first experiment was aimed at learning the definitions of ancestor, father and mother from the complete set of positive and negative examples and from the knowledge base containing male, female and parent as specified in Example 6. The system learned the following clauses in the specified order:

(1) ancestor(X,Y) ← parent(X,Y)
(2) father(X,Y) ← parent(X,Y), male(X)
(3) mother(X,Y) ← parent(X,Y), female(X)
(4) ancestor(X,Y) ← parent(X,Z), ancestor(Z,Y)

To check the ability of the MPL algorithm to recover from overgeneralization, we also ran MPL starting from clauses 5, 6 and 7.

(5) ancestor(X,Y) ← parent(X,Y)
(6) father(X,Y) ← ancestor(X,Y), male(X)
(7) mother(X,Y) ← ancestor(X,Y), female(X)
(8) ancestor(X,Y) ← parent(X,Z), parent(Z,Y)

MPL then discovered that clause 8 was the most interesting one to add next, yielding the current hypothesis $H$ composed of clauses 5, 6, 7 and 8: At that moment, the system discovered that the hypothesis is no longer globally consistent and it decided to remove (and mark) clauses 6 and 7 from $H$. After this, clauses 2, 3 and 4 were induced, making clause 8 redundant. Therefore clause 8 was removed, yielding the desired target theory.

### Experiment 2

The second experiment was aimed at learning male_ancestor and female_ancestor from father and mother. All the facts and examples about father, mother, male_ancestor and female_ancestor were derived from the background theory in Example 6. Experiment 2 is the same as Experiment B using FOIL. MPL learned (notice that clause 14 is correct) the following clauses:

(9) female_ancestor(X,Y) ← mother(X,Y)
(10) male_ancestor(X,Y) ← father(X,Y)
(11) female_ancestor(X,Y) ← mother(X,Z), female_ancestor(Z,Y)
(12) male_ancestor(X,Y) ← father(X,Z), female_ancestor(Z,Y)
(13) male_ancestor(X,Y) ← father(X,Z), male_ancestor(Z,Y)
(14) female_ancestor(X,Y) ← female_ancestor(X,Z), male_ancestor(Z,Y)

**Experiment 3**

In the third experiment the aim was to learn father and grandfather from an incomplete example set derived from the theory in Example 6. The example set contained all positive examples for father and grandfather. The negative examples were complete for grandfather and incomplete for father: $E^-_{\text{father}} = \{$ ← father(X,Y) | not parent(X,Y)$\}$. MPL derived the following globally complete and consistent theory:

father(X,Y) ← parent(X,Y)
grandfather(X,Y) ← male(X), parent(X,Z), parent(Z,Y)

Both GOLEM and FOIL are mislead on this example. They derive:

father(X,Y) ← parent(X,Y)
grandfather(X,Y) ← father(X,Z), parent(Z,Y)

This is globally complete but inconsistent. MPL does not synthesize the correct target theory either. Nevertheless, on this task, it does better than GOLEM and FOIL as it computes a solution to the *mpl* problem, which is not the case for GOLEM and FOIL.

## 7.3 Experiments with CLAUDIEN

Using a prototype implementation of CLAUDIEN in Prolog, two experiments were performed. In comparison with the FOIL and MPL results, which present the induced predicate definitions for multiple predicates, CLAUDIEN induces properties that hold in the given knowledge base.

Notice that the bias of CLAUDIEN was set such that only definite clauses were allowed with at most 2 literals in the body, with target predicates in the head and all logically redundant clauses were automatically ommited from the final hypothesis. The given bias was selected in order to be able to compare the results of the non-monotonic ILP setting using CLAUDIEN as much as possible with the results of the strong ILP setting as employed in MPL. Using a different bias a different set of clauses would have been induced, e.g. clauses with an empty head such as ← father(X,Y),mother(X,Y) in Experiment 1. Furthermore, as algorithms in the non-monotonic setting search the complete space of clauses,

a finite language bias is needed. Without a finite language bias, the algorithms would not terminate.

**Experiment 1**

Again, the first experiment was aimed at learning the definitions of ancestor, father and mother from the complete set of positive examples, and from the knowledge base containing male, female and parent as specified in Example 6. Notice that no negatives were given to CLAUDIEN, as CLAUDIEN assumes everything that is not positive to be negative. CLAUDIEN generated the following definite clauses:

(1) father( X, Y) ← parent( X, Y), father( X , Z)
(2) father( X, Y) ← parent( X, Y), male(X)
(3) mother( X, Y) ← parent( X, Y), mother( X , Z)
(4) mother( X, Y) ← parent( X, Y), female( X)
(5) ancestor( X, Y) ← parent( X, Y)
(6) ancestor( X, Y) ← father( X, Y)
(7) ancestor( X, Y) ← mother( X, Y)
(8) ancestor( X, Y) ← ancestor( X, Z), ancestor( Z, Y)

The reader may notice that the induced hypothesis is completely correct although it contains a number of clauses which are redundant. In particular, clauses (1,3,6,7) would normally not be generated in the normal setting. Clauses (6,7) would not be generate by CLAUDIEN if it knew that father implies parent and mother implies parent, and clauses (1,3) would not be generated if father implies male and mother implies female were known. In the absence of this knowledge, the clauses (1,3,6,7) do contain useful information. They do not only hold on the data, but they are useful for making predictions. Indeed, assume it is known that father(jef,paul) and that parent(jef,mary), but it is not known that male(jef). Using clause (1) one can deduce father(jef,mary), which is not possible without this clause. Therefore these clauses are also potentially useful.

**Experiment 2**

The second experiment was aimed at learning male_ancestor and female_ancestor from father and mother. All the facts and examples about father, mother, male_ancestor and female_ancestor were derived from the background theory in Example 6. CLAUDIEN generated the following clauses (using the same bias as in Experiment 1):

(1) male-ancestor( X, Y) ← male-ancestor( X, Z), female-ancestor( Z, Y)
(2) male-ancestor( X, Y) ← male-ancestor( X, Z), male-ancestor( Z, Y)
(3) female-ancestor( X, Y) ← female-ancestor( X, Z), female-ancestor( Z, Y)

(4) female-ancestor( X, Y) ← male-ancestor( Z, Y), female-ancestor( X, Z)
(5) male-ancestor( X, Y) ← father( X, Y)
(6) female-ancestor( X, Y) ← mother( X, Y)

In this case a completely (and non redundant) correct hypothesis was discovered.

### 7.4 Comments and conclusions

The experiments reported in this section, by large confirm the theoretical analysis of the previous section. In particular, the experiments with FOIL reveal the problems with ordering and extensional coverage; the MPL experiments confirm that a heuristic approach may overcome some of these limitations; and the CLAUDIEN experiments illustrate an alternative but nonetheless effective approach to learn multiple predicates.

The third experiment done with MPL was not repeated using CLAUDIEN, because the assumptions underlying CLAUDIEN were not satisfied. The example set MPL started from in this experiment was clearly incomplete, thereby invalidating the CLAUDIEN framework. This shows one of the drawbacks of the non-monotonic setting. When the closed world assumption on the examples is invalid, the non-monotonic setting would yield counter intuitive and incorrect results because it does not generalize on the examples, cf. Section 2. This makes it also impossible to apply the non-monotonic setting to program synthesis from (incomplete) example sets.

## 8 Conclusions

In the first part of this paper we defined and investigated the different faces of ILP, focussing on the two different ILP semantics: the normal and non-monotonic semantics. The two semantics were illustrated on a knowledge discovery problem and on a program synthesis problem.

The second part of the paper focussed on the use of the two semantics in the context of multiple predicate learning. Examining the problem specification of the normal ILP setting reveals that the clauses in a hypothesis are not independent of each other. Indeed, consider clauses $c_1$ and $c_2$. Assume that $B \cup \{c_i\}$ $(i = 1..2)$ imply some positive and no negative examples. So, both $c_i$ could contribute to a solution in the normal setting. Therefore one might consider a hypothesis $H$ containing $\{c_1, c_2\}$. Although neither $c_1$ nor $c_2$ entails negative examples, it may be that their conjunction $B \cup \{c_1, c_2\}$ does. This reveals that in the normal ILP setting different clauses of a hypothesis may interact. Because of these interactions, learning multiple predicates in the normal ILP setting is necessarily order dependent as the coverage of one clause depends on the other clauses in the hypothesis. Therefore the order of learning different clauses affects the results of the learning task and even the existence of solutions. As

an extreme case, a multiple predicate learning task may only be solvable when clauses (or predicates) are learned in a specific order (because of bias effects and interactions between clauses). In less extreme situations, certain orders may result in more complex (and imprecise) definitions whereas better orders may yield good results.

Present ILP learners in the normal setting provide little help in determining the next clause or predicate to be learned. Indeed, the incremental systems (e.g. MOBAL [20]) rely on the user as they treat the examples in the given order, the interactive ones (CIGOL [32], CLINT [8], and MIS [42]) also generate queries to further simplify the problem, whereas the empirical ones (such as GOLEM [34] and FOIL [40]) leave the problem to the user. For instance, the FOIL algorithm outlined in Algorithm 1 checks only that individual clauses are consistent, not that the hypothesis as a whole is consistent with the negative examples. An attempt to alleviate the above problems is incorporated in the MPL system [14] which solves many of the problems encountered in multiple predicate learning. Still, one major problem remains: when generating new clauses, although neither $c_1$ nor $c_2$ (constituting $H$) entails negative examples, it may be that their conjunction $B \cup \{c_1, c_2\}$ does. Thus backtracking is required, i.e. already generated clauses must be re-evaluated and potentially deleted since as a part of the currently generated hypothesis they may become inconsistent. Due to the computational complexity of backtracking we may consider the normal ILP setting as inappropriate for multiple predicate learning.

We believe the non-monotonic ILP semantics puts several issues in ILP in a new perspective. First, the non-monotonic setting allows to induce full clausal theories. Second, it makes the learning of different clauses order independent, which is especially useful when learning multiple predicates. Third, learning in the non-monotonic setting derives properties of examples instead of rules generating examples. Although such properties cannot always be used for predicting the truthvalues of facts, we have seen that the use of full clausal theories can result in predictions not possible using the less expressive normal ILP setting. However, when we are interested in predictions, then the normal setting is more appropriate as induced hypotheses in the normal setting can always be used for prediction. Also, when the example set is incomplete (the closed world assumption does not hold), the non-monotonic setting cannot be applied. This explains why the non-monotonic framework does not apply to program synthesis from examples. Although learning multiple predicates in the normal setting is less efficient, the approach to multiple predicate learning in the normal setting, implemented in the MPL algorithm, can sometimes alleviate the problems encountered by the standard ILP systems.

Finally, let us mention that there is still a third possible setting for inductive logic programming proposed by Flach [18]. In this setting, one is looking for hypotheses $H$ that are consistent with the examples, i.e. given $B$ and $E$, find $H$ such that $B \cup H \cup E \not\models \square$. This condition is related to the consistency requirement in the normal setting except that $E$ can include positive as well as

negatives here. Therefore it follows that also in Flach's setting, clauses are not to be searched independenty from each other. Furthermore, Flach's setting is even more permissive than the normal one in the sense that if $H$ is a solution in the normal setting, it will also be a solution in Flach's setting.


## Acknowledgements

This work is part of the ESPRIT Basic Research project no. 6020 on Inductive Logic Programming. Luc De Raedt is supported by the Belgian National Fund for Scientific Research; Nada Lavrač is funded by the Slovenian Ministry of Science and Technology. The authors are grateful to Sašo Džeroski for the experiments with FOIL, and to Danny De Schreye, Bern Martens, Stephen Muggleton and Gunther Sablon for discussions, suggestions and encouragements concerning this work. Special thanks to Maurice Bruynooghe, Peter Flach and the referees for suggesting many improvements to earlier versions of this paper.


# References


[1] H. Adé, L. De Raedt, and M. Bruynooghe. Declarative Bias for Specific-To-General ILP Systems. *Machine Learning*, to appear, 1995.

[2] F. Bergadano and D. Gunetti. An interactive system to learn functional logic programs. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1044–1049. Morgan Kaufmann, 1993.

[3] I. Bratko. Applications of machine learning : towards knowledge synthesis. In *Proceedings of Future Generation Computing Systems*, 1992.

[4] Wray Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36:375–399, 1988.

[5] R.M. Cameron-Jones and J.R. Quinlan. Avoiding pitfalls when learning recursive theories. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1050–1055. Morgan Kaufmann, 1993.

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

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

[8] L. De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press, 1992.

[9] L. De Raedt and M. Bruynooghe. Belief updating from integrity constraints and queries. *Artificial Intelligence*, 53:291–307, 1992.

[10] L. De Raedt and M. Bruynooghe. Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150, 1992.

[11] L. De Raedt and M. Bruynooghe. A unifying framework for concept-learning algorithms. *The Knowledge Engineering Review*, 7(3):251–269, 1992.

[12] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann, 1993.

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

[14] L. De Raedt, N. Lavrač, and S. Džeroski. Multiple predicate learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1037–1042. Morgan Kaufmann, 1993.

[15] T.G. Dietterich and R.S. Michalski. Learning to predict sequences. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume 2. Morgan Kaufmann, 1986.

[16] B. Dolšak and S. Muggleton. The application of Inductive Logic Programming to finite element mesh design. In S. Muggleton, editor, *Inductive logic programming*, pages 453–472. Academic Press, 1992.

[17] S. Džeroski and I. Bratko. Handling noise in inductive logic programming. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, 1992.

[18] P. Flach. A framework for inductive logic programming. In S. Muggleton, editor, *Inductive logic programming*. Academic Press, 1992.

[19] N. Helft. Induction as nonmonotonic inference. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 149–156. Morgan Kaufmann, 1989.

[20] 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.

[21] R.D. King, S. Muggleton, R.A. Lewis, and M.J.E. Sternberg. Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 89(23), 1992.

[22] R. Korf. Depth-first iterative deepening : an optimal admissable search. *Artificial Intelligence*, 27:97–109, 1985.

[23] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[24] N. Lavrač, S. Džeroski, and M. Grobelnik. Learning non-recursive definitions of relations with LINUS. In Yves Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1991.

[25] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 2nd edition, 1987.

[26] R.S. Michalski. A theory and methodology of inductive learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.

[27] T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[28] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–317, 1991.

[29] S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

[30] S. Muggleton. Predicate invention and utility. *Journal for Experimental and Theoretical Artificial Intelligence*, 1994. To appear.

[31] S. Muggleton, M. Bain, J. Hayes-Michie, and D. Michie. An experimental comparison of human and machine learning formalisms. In *Proceedings of the 6th International Workshop on Machine Learning*, pages 113–118. Morgan Kaufmann, 1989.

[32] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning*, pages 339–351. Morgan Kaufmann, 1988.

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

[34] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st conference on algorithmic learning theory*, pages 368–381. Ohmsma, Tokyo, Japan, 1990.

[35] S. Muggleton, R.D. King, and M.J.E. Sternberg. Protein secondary structure prediction using logic. *Protein Engineering*, 7:647–657, 1992.

[36] T. Niblett. A study of generalisation in logic programs. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning*, pages 131–138. Pitman, 1988.

[37] G. Piatetsky-Shapiro and W. Frawley, editors. *Knowledge discovery in databases*. The MIT Press, 1991.

[38] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

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

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

[41] S.J. Russell. *The use of knowledge in analogy and induction*. Pitman, 1989.

[42] E.Y. Shapiro. *Algorithmic Program Debugging*. The MIT Press, 1983.

[43] M.E. Stickel. A prolog technology theorem prover: implementation by an extended prolog compiler. *Journal of Automated Reasoning*, 4(4):353–380, 1988.

[44] R. Wirth. Learning by failure to prove. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning*. Pitman, 1988.

[45] R. Wirth. Completing logic programs by inverse resolution. In K. Morik, editor, *Proceedings of the 4th European Working Session on Learning*. Pitman, 1989.

[46] S. Wrobel. Automatic representation adjustment in an observational discovery system. In Sleeman D., editor, *Proceedings of the 3rd European Working Session on Learning*. Pitman, 1988.