

First Order jk -Clausal Theories are PAC-Learnable*

Luc De Raedt

Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
Luc.DeRaedt@cs.kuleuven.ac.be

Sašo Džeroski

Artificial Intelligence Laboratory, Jožef Stefan Institute
Jamova 39, 61111 Ljubljana, Slovenia
Email: Saso.Dzeroski@ijs.si

Abstract

We present positive PAC-learning results for the nonmonotonic inductive logic programming setting. In particular, we show that first order range-restricted clausal theories that consist of clauses with up to k literals of size at most j each are polynomial-sample polynomial-time PAC-learnable with one-sided error from positive examples only. In our framework, concepts are clausal theories and examples are finite interpretations. We discuss the problems encountered when learning theories which only have infinite non-trivial models and propose a way to avoid these problems using a representation change called flattening. Finally, we compare our results to PAC-learnability results for the normal inductive logic programming setting.

1 Introduction

Recently, both inductive logic programming (Muggleton and De Raedt 94, Muggleton 92), which studies the induction of first order logical formulae from examples and background knowledge, and computational learning theory (Valiant 84, Natarajan 91), which is concerned with the convergence and complexity of learning algorithms, have received a lot of attention. It is therefore no surprise that several researchers have started to investigate the computational properties of inductive logic programming (Page and Frisch, 92, Džeroski et al. 92, 93, Cohen 93ab, Kietz 93, Page 93, Kietz and Džeroski 94). However, to the best of our knowledge, all such results hold for the normal inductive logic programming setting, where the aim is to find a hypothesis H starting from a background theory B and positive and negative examples P and N such that $B \wedge H \models P$ and $B \wedge H \wedge N \not\models \square$ (cf. Muggleton

*First version January 1994, second and revised version May 1994.

and De Raedt 94). Furthermore, some of the results by Kietz (93) and Cohen (93ab) are negative and show that many interesting learning tasks are not tractable, i.e. not polynomially solvable.

At present, however, there is a growing interest (cf. Muggleton and De Raedt 94, De Raedt and Bruynooghe 93, De Raedt and Lavrač 93, Flach 92) in the so-called nonmonotonic inductive logic programming setting, first introduced by Helft (89). The main difference between the normal and the nonmonotonic inductive logic programming setting is that the former setting is interested in classification rules, whereas the latter searches for properties of the examples, cf. (De Raedt and Lavrač 93, Muggleton and De Raedt 94).

In this paper, we formalize the nonmonotonic inductive logic programming setting, in which concepts are clausal theories and examples are finite interpretations. Interpretations that are models of the theory are positive examples. The main result of this paper is that range-restricted clausal theories consisting of clauses with up to k literals of size at most j each are polynomial-sample polynomial-time PAC-learnable with one-sided error from positive examples only. This class of clausal theories is called jk -CT. The problems encountered when learning theories which only have infinite non-trivial models are also discussed, together with an effective way of handling them. Finally, we relate our results to PAC-learnability results for the normal inductive logic programming setting.

The paper is structured as follows: Section 2 introduces the nonmonotonic inductive logic programming setting. Section 3 introduces the PAC-learning framework. In Section 4, we prove our main learnability result. Section 5 discusses problems encountered when learning theories which only have infinite non-trivial models, and introduces the flattening transformation, which can be used to avoid these problems. Finally, Section 6 discusses related work and concludes.

2 Non-monotonic inductive logic programming

We first outline some standard logic programming concepts (see (Lloyd 87) for more details).

A first order alphabet is a set of predicate symbols and a set of functor symbols. A clause is a formula of the form $A_1, \dots, A_m \leftarrow B_1, \dots, B_n$, where the A_i and B_i are logical atoms. The A_i are also sometimes called positive literals, and the B_j negative literals. An atom $p(t_1, \dots, t_n)$ is a predicate symbol p followed by a bracketed n -tuple of terms t_i . A term t is a variable V or a functor symbol $f(t_1, \dots, t_k)$ immediately followed by a bracketed n -tuple of terms t_i . Constants are functor symbols of arity 0. *Functor free* clauses are clauses that contain only variables as terms.

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. Extending the usual convention for *definite clauses* (where $m = 1$), we call A_1, \dots, A_m the *head* of the clause and B_1, \dots, B_n the *body* of the clause. A *fact* is a definite clause with an empty body, ($m = 1, n = 0$). A clausal theory, or shortly theory, is a set of clauses. The set of variables in a term, atom or clause e , is denoted by $vars(e)$. Throughout the paper, we shall assume that all clauses are *range restricted*, which means that all variables occurring in the head of a clause also occur in its body, i.e. $vars(head(c)) \subseteq vars(body(c))$.

The size of a term, atom, clause, or theory, is the number of symbols in it, i.e. the total

number of occurrences of all predicate symbols, functor symbols, and variables. By now we are able to define the class of theories considered:

Definition 1 *jk-CT, the class of jk-clausal theories, is the class of all theories composed of range-restricted clauses that contain at most k literals per clause, where each literal (atom) is of size at most j.*

A substitution $\theta = \{V_1/t_1, \dots, V_n/t_n\}$ is an assignment of terms t_i to variables V_i . Applying a substitution θ to a term, atom, or clause e yields the instantiated term, atom, or clause $e\theta$ where all occurrences of the variables V_i are simultaneously replaced by the terms t_i . A term, atom or clause e is called ground when there is no variable occurring in e , i.e. $vars(e) = \emptyset$. A *Herbrand interpretation* over a first order alphabet is a set of ground facts constructed with the predicate and functor symbols in the alphabet. A Herbrand interpretation I is a model for a clause c if and only if for all substitutions θ such that $c\theta$ is ground: $body(c)\theta \subset I \rightarrow head(c)\theta \cap I \neq \emptyset$. We also say c is true in I . A Herbrand interpretation I is a model for a clausal theory T if and only if it is a model for all clauses in T . Roughly speaking, the truth of a (range-restricted) clause c in a (finite) interpretation I can be determined by running the query $? - body(c), not head(c)$ on a database containing I using a PROLOG system. If the query succeeds, the clause is false in I . If the query fails, the clause is true.

Example 1 *Consider theory T consisting of the following two clauses:*

$c_1 = parent(father(X), X) \leftarrow human(X)$ and $c_2 = parent(mother(X), X) \leftarrow human(X)$.

Let $I_1 = \{human(jef), parent(father(jef), jef), parent(mother(jef), jef)\}$. I_1 is a model for T . On the other hand, $I_2 = I_1 \cup \{human(mary)\}$ is not a model for T as there exists a substitution $\theta = \{X/mary\}$ such that $c_1\theta$ is false in I_2 , as $body(c_1)\theta \subset I_2$ and $head(c_1)\theta \cap I_2 = \emptyset$. Intuitively, the theory T is a property of interpretation I_1 , but not of interpretation I_2 .

By now we can formalize the nonmonotonic inductive logic programming setting. In the nonmonotonic setting, a set of ground facts (model) is given, as well as a class (set) of clauses \mathcal{C} . The aim is to find a maximal hypothesis (i.e. a maximal subset of \mathcal{C}) that is true in the model¹. Such hypotheses reflect true properties of the model, i.e. they hold on the observations. In the normal inductive logic programming setting, however, one is mainly interested in prediction or classification, and learns from positive as well as negative examples. In the nonmonotonic setting, there are no negative examples as one makes a kind of closed world assumption, and assumes that everything not stated true is false. For a full discussion of the relation between the two settings, we refer to (Muggleton and De Raedt 94, De Raedt and Bruynooghe 93, De Raedt and Lavrač 93). The nonmonotonic setting is illustrated in Example 2.

Example 2 *Let $M_1 = \{ mother(lieve, soetkin), father(luc, soetkin), parent(lieve, soetkin), parent(luc, soetkin), male(luc), female(lieve), female(soetkin), human(lieve), human(luc), human(soetkin) \}$.*

¹Usually, not only truth is required, but also non-redundancy, cf. (Muggleton and De Raedt 94, De Raedt and Bruynooghe 93, De Raedt and Lavrač 93).

Consider $W = \{ \text{parent}(X, Y) \leftarrow \text{mother}(X, Y); \text{parent}(X, Y) \leftarrow \text{father}(X, Y); \text{mother}(X, Y) \vee \text{father}(X, Y) \leftarrow \text{parent}(X, Y); \leftarrow \text{mother}(X, Y) \wedge \text{father}(X, Y); \text{human}(X) \leftarrow \text{female}(X); \text{human}(X) \leftarrow \text{male}(X); \text{female}(X) \vee \text{male}(X) \leftarrow \text{human}(X); \leftarrow \text{female}(X) \wedge \text{male}(X); \text{female}(X) \leftarrow \text{mother}(X, Y); \text{male}(X) \leftarrow \text{father}(X, Y); \text{human}(X) \leftarrow \text{parent}(X, Y); \text{human}(Y) \leftarrow \text{parent}(X, Y); \leftarrow \text{parent}(X, X) \}$

Here, M_1 is a model for W or W is true in M_1 . The same holds if we replace M_1 by $M_2 = \{ \text{mother}(\text{blaguna}, \text{sonja}), \text{father}(\text{veljo}, \text{saso}), \text{father}(\text{veljo}, \text{sonja}), \text{parent}(\text{blaguna}, \text{saso}), \text{parent}(\text{blaguna}, \text{sonja}), \text{parent}(\text{veljo}, \text{saso}), \text{parent}(\text{veljo}, \text{sonja}), \text{male}(\text{veljo}), \text{male}(\text{saso}), \text{female}(\text{blaguna}), \text{female}(\text{sonja}), \text{human}(\text{veljo}), \text{human}(\text{saso}), \text{human}(\text{blaguna}), \text{human}(\text{sonja}) \}$

Theory W is a functor free theory where $j = 3$ and $k = 3$, hence it is in 3,3-CT. Theory W (or equivalent ones) could be induced by the inductive logic programming system CLAUDIEN (De Raedt and Bruynooghe 93) starting from 3,3-CT and M_1, M_2 or $M_1 \cup M_2$.

We illustrate the nonmonotonic learning setting by a simple example. Suppose we are making a poll of family relations in Europe with the aim of discovering the properties of such relations. To realize our aim, we could take a European phone book, phone people and ask them for facts about their family. Each phone-call would result in one particular model (such as M_1 and M_2) of the target theory (e.g. W) which would contain all properties of European families.

Formulated differently, the target theory (the concept) divides the set of all finite Herbrand interpretations in two subsets : the models of the target theory (i.e. the positive examples) and the interpretations of the target theory that are false (i.e. the negative examples). Given this viewpoint, learning clausal theories in the nonmonotonic inductive logic programming setting corresponds to a classical concept-learning task. The main difference is that the examples are true and false interpretations of the target theory. Furthermore, in this setting (as in the poll-scenario) it is natural to learn from positive examples (models) only. Notice that this setting naturally extends the representation formalism used in propositional learning systems, where each example is described by a set of propositions, i.e. facts about nullary predicates. In our nonmonotonic setting, each example is described by a set of general facts. Because the arity of the predicates is not limited to 0, an example can contain multiple facts per single predicate.

3 PAC-learning

We first formalize the PAC-learning paradigm introduced by Valiant (84), (closely following Natarajan (91)), and then apply it to the nonmonotonic ILP setting.

Definition 2 *A concept is a subset of a universal set of objects U . A class of concepts is a set of concepts, i.e. a subset of 2^U , the power set of U . An object e is a positive example for a concept C if $e \in C$ and a negative example otherwise.*

Let \mathcal{F} be a class of concepts. The target concept f may be any concept in \mathcal{F} . A learning algorithm for \mathcal{F} is an algorithm that attempts to construct an approximation to the target

concept from examples for it. The learning algorithm takes as input two parameters: the error parameter $\epsilon \in (0, 1]$ and the confidence parameter $\delta \in (0, 1]$. The error parameter specifies the error allowed in a good approximation and the confidence parameter controls the likelihood of constructing a good approximation.

The learning algorithm has at its disposal a subroutine *EXAMPLE*, which at each call produces a single example for the target concept f . The probability that a particular example $e \in U$ (positive or negative for f) will be produced at a call of *EXAMPLE* is $D(e)$, where D is an arbitrary and unknown distribution on U . The choice of the distribution D is independent of the target concept f .

Concept g is a good approximation of concept f if the probability that f and g differ on a randomly chosen example from U is at most ϵ , i.e. $D(f\Delta g) \leq \epsilon$, where $f\Delta g = f - g \cup g - f$. Putting all of the above together, we obtain the following definition.

Definition 3 *An algorithm A is a probably approximately correct (PAC) learning algorithm for a class of concepts \mathcal{F} if*

1. *A takes as input $\epsilon \in (0, 1]$ and $\delta \in (0, 1]$.*
2. *A calls *EXAMPLE*, which returns examples for some $f \in \mathcal{F}$. The examples are chosen randomly according to an arbitrary and unknown probability distribution D on U .*
3. *For all concepts $f \in \mathcal{F}$ and all probability distributions D on U , A outputs a concept $g \in \mathcal{F}$, such that with probability at least $(1 - \delta)$, $D(f\Delta g) \leq \epsilon$.*

A class \mathcal{F} is PAC-learnable if there exists an algorithm A which is a PAC-learning algorithm for \mathcal{F} .

Definition 4 *Algorithm A learns \mathcal{F} with one-sided error if A is a PAC-learning algorithm for \mathcal{F} and the concept output by A is always a subset of the target concept.*

To illustrate the above definitions, consider the following result about k -CNF. A k -CNF formula corresponds to a jk -CT theory where all predicates are of arity 0, i.e. propositional. Therefore, the size of each atom is exactly one. Usually, however, one defines a k -CNF formula as a CNF (conjunctive normal form) formula, where each disjunction (clause) consists of at most k literals (Boolean variables or their negations). For example, the formula $f = (x_1 \vee \bar{x}_2) \wedge x_4 \wedge (\bar{x}_2 \vee x_3)$ is a 2-CNF formula over four Boolean variables. Written in jk -CT notation, we have $f = \{x_1 \leftarrow x_2, x_4 \leftarrow, x_3 \leftarrow x_2\}$ which is a $1k$ -CT theory with four propositional predicates. The class of k -CNF is known to be PAC-learnable with one-sided error (Valiant 84) from positive examples only.

Our nonmonotonic inductive logic programming problem can be formulated in the above PAC-framework as follows:

- The set of objects (examples) U is the set of all finite Herbrand interpretations.
- \mathcal{F} , the class of concepts studied, is the class of jk -clausal theories.
- *EXAMPLE* = *MODEL*, i.e. each call to the oracle produces a model of the target theory W . This implies that we are learning from positive examples only.

Without loss of generality, we will assume that the set of predicate symbols P , and the set of functor symbols F , to be used in the theory and its models, are given initially. The size of the problem will be characterized by the sizes of P and F , as well as with m , the size of the largest example model. Thus, m is the largest number of symbols (occurrences of predicates, functors and variables) in any single model (or positive example).

Definition 5 *Let A be a learning algorithm for a class of clausal theories CT over P and F . The sample complexity of A is the function $s:\mathbf{R}\times\mathbf{R}\times\mathbf{N}\times\mathbf{N}\rightarrow\mathbf{N}$, such that $s(\epsilon, \delta, |P|, |F|)$ is the maximum number of calls of MODEL by A , the maximum being taken over all runs of A on inputs ϵ, δ , with the target concept ranging over all concepts in CT and the probability distribution D ranging over all distributions on U . If no finite maximum exists, $s(\epsilon, \delta, |P|, |F|) = \infty$. A class CT is said to be polynomial-sample PAC-learnable if there exists a PAC-learning algorithm A for CT with sample complexity $p(1/\epsilon, 1/\delta, |P|, |F|)$, where p is a polynomial function of its arguments.*

Definition 6 *Let A be a learning algorithm for a class of clausal theories CT over P and F . The time complexity of A is the function $t:\mathbf{R}\times\mathbf{R}\times\mathbf{N}\times\mathbf{N}\times\mathbf{N}\rightarrow\mathbf{N}$, such that $t(\epsilon, \delta, |P|, |F|, m)$ is the maximum number of computational steps consumed by A , the maximum being taken over all runs of A on inputs ϵ, δ , during which m is the size of the largest example seen by A , with the target concept ranging over all concepts in CT , and the probability distribution D ranging over all distributions on U . If no finite maximum exists, $t(\epsilon, \delta, |P|, |F|, m) = \infty$. A class CT is said to be polynomial-time PAC-learnable if there exists a PAC-learning algorithm A for CT with time complexity $p(1/\epsilon, 1/\delta, |P|, |F|, m)$, where p is a polynomial function of its arguments.*

4 Learnability result

The **Learn-Clausal-Theory** algorithm, given in Figure 1, takes as input the maximum atom size j , the maximum number of literals per clause k , the sets of predicate and functor symbols P and F , and the PAC performance parameters ϵ and δ . The number of variables that can be used in the clauses is implicitly limited by $j \times k$. Namely, as each clause can have at most k literals, and each literal at most j variables, there can be at most jk variables in a clause. Thus, we will assume $V = \{V_1, V_2, \dots, V_{jk}\}$.

The algorithm attempts to find a good approximation T of a target k -literal clausal theory W . It has at its disposal a subroutine MODEL, analogous to the subroutine EXAMPLE mentioned in Section 3. Upon calling, MODEL returns a finite set M of ground facts for the predicates from P , such that W is true in M , i.e. M is a model for W . An arbitrary probability distribution D is assumed over the set of finite Herbrand models for W .

Theorem 1 *The algorithm **Learn-Clausal-Theory** is a PAC-learning algorithm for the class jk -CT, has polynomial sample and time complexity, and learns with one-sided error.*

Proof:

Learn-Clausal-Theory($j, k, P, F, \epsilon, \delta$)

1. Construct a set V that contains $j \times k$ different variables
 2. $T_0 := \{C \mid C \text{ is range-restricted and } C = h_1 \vee \dots \vee h_r \leftarrow g_1 \wedge \dots \wedge g_s, \text{ where } r + s < k + 1 \text{ and all of the } h_i \text{ and } g_i \text{ are atoms of size at most } j, \text{ constructed of symbols from } P, F \text{ and } V\}$.
 3. $T := T_0$
 4. **for** $i := 1$ **to** $\max\left(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{8 \times |T_0|}{\epsilon} \log \frac{13}{\epsilon}\right)$ **do**
 - $M := \text{MODEL}$
 - **foreach** $C \in T$ **do**
if C is false in M **then** delete C from T
 5. **return** T
-

Figure 1: A PAC-learning algorithm for jk -clausal theories.

1. **Learn-Clausal-Theory** is a PAC-learning algorithm for jk -CT and has polynomial sample complexity.

This result follows from Theorem 2.1 by Blumer et al. (89) provided that

- (a) **Learn-Clausal-Theory** outputs a consistent hypothesis T , and
- (b) jk -CT is of polynomial VC-dimension. The corresponding sample-complexity is then

$$\max\left(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{8 \times \text{VC-dim}(jk\text{-CT})}{\epsilon} \log \frac{13}{\epsilon}\right).$$

a. Consistency

To see that T is consistent with the observed examples, notice that the clauses in T are true in each of the example models, as for each example M clauses that are false in M have been deleted from T .

b. VC-dim(jk -CT) is polynomial

Observe that $\text{VC-dim}(jk\text{-CT}) \leq \log |jk\text{-CT}|$, see e.g. (Natarajan 91). Knowing that $jk\text{-CT} = 2^{T_0}$, we can conclude that $\text{VC-dim}(jk\text{-CT}) \leq |T_0|$. It now remains to be proven that $|T_0|$ is polynomial in $|P|$ and $|F|$.

Recall that T_0 is the set of all range restricted clauses with at most k literals of size at most j each. Observe that $|T_0| < 1 + 2a + (2a)^2 + \dots + (2a)^k < (2a)^{k+1}$, where a is the number of atoms of size at most j using symbols from P, F and V . We now prove that $a < |P|(|F| + jk)^j$. In an atom of size i , exactly $i - 1$ symbols from F and V appear. These can be placed in at most $(|F| + |V|)^{i-1} = (|F| + jk)^{i-1}$ ways, as in a literal of size i there are exactly $i - 1$ places for functors and variables, regardless of their level of nesting.

Thus there are at most $|P|(|F| + jk)^{i-1}$ different atoms of size i . Summing over all possible sizes, we yield $a < |P| \sum_{i=1}^j (|F| + jk)^{i-1} < |P|(|F| + jk)^j$.

2. **Learn-Clausal-Theory** has polynomial time complexity.

As there is only a polynomial number of examples and clauses that is processed, it suffices to prove that the truth of a clause C in a model or example M can be determined in polynomial time (in the size of the example m). This can be done using the following procedure: find all substitutions θ such that $body(C)\theta \subset M$ and check whether $head(C)\theta \cap M \neq \emptyset$. The substitutions θ for which $body(C)\theta$ is true, can be found by (repeatedly) deleting a literal from $body(C)$ and unifying that literal in all possible ways with literals in M . Unifying two literals, one of which is ground, can clearly be done in time linear in the size of the ground atoms. Therefore the substitutions for one literal from $body(C)$ can be found in time $O(m^2)$, and as there are at most k literals in C , the truth of C in M can be determined in $O(m^{2k})$. Therefore, the algorithm **Learn-Clausal-Theory** has polynomial time complexity.

3. **Learn-Clausal-Theory** learns jk -CT with one-sided error.

As we have already proved that **Learn-Clausal-Theory** is a PAC-learning algorithm for jk -CT, we now show that the concept identified by the algorithm is always a subset of the target concept. We do so by proving that as the algorithm proceeds $\mathcal{T} \subset \mathcal{W}$, where \mathcal{T} (respectively \mathcal{W}) is the set of interpretations in which T (respectively W) is true.

Initially, $\mathcal{T} = \emptyset$, as $T = T_0$, and the empty (inconsistent) clause belongs to T_0 .

Let $B = \{c \in T_0 \mid \forall M \in U : \text{if } W \text{ is true in } M \text{ then } c \text{ is true in } M\}$. At all stages of the algorithm $B \subset T$, and therefore $\mathcal{T} \subset \mathcal{B}$, where \mathcal{B} is the set of interpretations in which B is true. Namely, no clause in B can be deleted from T in the course of the algorithm, as MODEL only returns examples M such that W is true in M .

To prove $\mathcal{T} \subset \mathcal{W}$, we show that $\mathcal{B} = \mathcal{W}$, i.e. $\models_U B \leftrightarrow W$ (for all interpretations over U , B is true if and only if W is true).

(a) $\models_U B \rightarrow W$ because $W \subset B$. To prove $W \subset B$, suppose some clause $c' \in W$ did not occur in B . c' would then have the property “ $\exists M$ such that W is true in M and c' is not true in M ”, which is a contradiction.

(b) $\models_U W \rightarrow B$. This follows directly from the definition of B . \square

Corollary 1 *The class jk -CT is polynomial-sample polynomial-time PAC-learnable with one-sided error from positive examples only.*

Corollary 2 *k -CNF is polynomial-sample polynomial-time PAC-learnable with one-sided error from positive examples only.*

Proof: Observe that k -CNF = $1k$ -CT. This is the original result by Valiant (84). \square

Note that jk -CT = $2^{C_{jk}}$, where C_{jk} is the set of all range restricted clauses with at most k literals of size at most j each. Our learnability result holds also for all classes of

clausal theories CT, such that $CT = 2^{C_l}$, where $C_l \subset C_{jk}$.² This is important, as much of the current efforts in inductive logic programming are devoted to building frameworks for declarative bias specification (cf. Cohen 94, Muggleton and De Raedt 94, Kietz and Wrobel 92, Bergadano 93, Ade et al. 94). By declarative bias specification, we mainly mean flexible formalisms for specifying the syntax of clauses in C_l , i.e. the clauses to be used in the induced theories. Proposed formalisms include the antecedent description grammars of Cohen, the rule-models of Kietz and Wrobel, the predicate sets of Bergadano and their integration by Ade et al. Having such a bias would significantly reduce the number of clauses considered and would result in more practical bounds on the time and sample complexity. In the implemented system CLAUDIEN of (De Raedt and Bruynooghe 93), the bias formalism of Ade et al. is employed.

5 Learning theories with infinite models

As our framework assumes finite models, one might expect problems when infinite models are involved. One such problem involves learning clausal theories with an infinite intended interpretation. This problem is not always significant as theories with an infinite intended interpretation may also have finite models. A second, more serious problem, arises when clauses have only infinite non-trivial models. A non-trivial model for a clause is a model for which there exist substitutions that make the body true. We will now discuss these problems and will show how flattening can be used to alleviate the second problem.

Example 3 *Suppose our target theory is $T = \{sort([A|B], S) \leftarrow partition(A, B, C, D), sort(C, E), sort(D, F), append(E, [B|F], S)\}$. The interpretation (set of ground facts) $I = \{append([1], [2], [1, 2]), partition(2, [1], [1], []), sort([], []), sort([1], [1]), sort([2, 1], [1, 2])\}$ is a model (i.e. a positive example) for the target theory T . We have used the PROLOG notation for lists here, where $[]$ stands for nil, the empty list, and $[X|Y]$ stands for $cons(X, Y)$ in LISP notation, where $cons$ is a function symbol.*

The intended interpretation of the target theory T is infinite, and contains all facts of the predicates $sort$, $append$ and $partition$ about lists consisting of the elements 1 and 2 (an infinite number of lists!) and possibly 3, 4, ... Nevertheless, I is a non-trivial finite model of T , as for each substitution θ for which $c\theta$ is ground, $body(c)\theta \subset I \rightarrow head(c)\theta \in I$. Still, this example clearly indicates that one should carefully select the examples when functors are present. However, we believe that this problem is not specific to our setting but also holds for the normal inductive logic programming setting. Indeed, ever since the work of Shapiro (1983) (and in particular his "lazy" approach, which is also followed in GOLEM (Muggleton and Feng 1990) and FOIL (Quinlan 1990)), it is well known that in order to learn recursive theories, one should construct the examples so that the recursive case can fall back on the base case. In practice this means that learning the recursive clause for member,

²To PAC-learn a concept class CT of the above form, the algorithm **Learn-Clausal-Theory** has to be changed to initialize T_0 to C_l (as stated now, T_0 is initialized to C_{jk}).

starting from an example such as $member(a, [b, c, a])$ will only succeed when $member(a, [a])$ is present ³.

A more serious problem arises when the target theory has no non-trivial finite models, e.g. when it includes a clause such as $nat(s(X)) \leftarrow nat(X)$. As our framework assumes finite models, the reader might believe that our framework cannot learn such clauses. ILP systems that work in the normal setting, however, have no problems with learning clauses similar to the above. Namely, some of them use λ functor-free representation and use a flattening (Rouveirol 92, 94) change of representation to deal with structured terms. We will now introduce a flattening transformation for clausal theories and interpretations.

Definition 7 *The flattened clause of a clause C is $flat_clause(C) = C$ if C is functor free, otherwise $flat_clause(C) = flat_clause(C_f)$, where C_f is obtained from C by substituting all occurrences of a non-variable term $f(t_1, \dots, t_n)$ by a variable V (not occurring in C) and adding $f_p(t_1, \dots, t_n, V)$ to the body of C . A unique flattening predicate symbol f_p is introduced for each functor f .*

Definition 8 *The unflattened clause of a functor free clause C is $unflat_clause(C) = C$ if C contains only regular predicates, otherwise $unflat_clause(C) = unflat_clause(C_u)$, where C_u is obtained by resolving C with $f_p(X_1, \dots, X_n, f(X_1, \dots, X_n)) \leftarrow$, where f_p occurs in C .*

When flattening clauses and interpretations each functor f of arity a is replaced by a unique flattening predicate f_p of arity $a + 1$, and compound terms in clauses are flattened by adding literals about the flattening predicates to the body of the clause. For instance, $flat_clause(p(f(f(a))) \leftarrow p(f(a))) = p(X) \leftarrow p(Z), f_p(Z, X), a_p(Y), f_p(Y, Z)$. Notice also that $unflat_clause(flat_clause(C)) = C$, cf. Rouveirol (92,94).

Whereas Rouveirol defines flattening only for clauses, we also need a flattening operation to transform interpretations into flattened interpretations⁴.

Definition 9 *The flattened interpretation $flat_int(I)$ of an interpretation I is*

$$\bigcup_{q(t_1, \dots, t_n) \in I} (\{q(c_{t_1}, \dots, c_{t_n})\} \cup \{f_p(c_{u_1}, \dots, c_{u_m}, c_{f(u_1, \dots, u_m)}) \mid f(u_1, \dots, u_m) \text{ is a subterm of some } t_i\}),$$

where each c_t denotes a unique constant.

Compound terms in an interpretation are flattened by adding ground facts about the flattening predicates, and by introducing unique constants c_t for each term appearing in the interpretation. To map the constants c_t back into the terms t , we use the mapping $unflat_sub$, which is defined below. Notice that this is a one-to-one mapping.

Definition 10 *The unflattened substitution $unflat_sub(\theta)$ of a substitution $\theta = \{X_1/c_{t_1}, \dots, X_n/c_{t_n}\}$ is $\{X_1/t_1, \dots, X_n/t_n\}$.*

³Alternatively, in the normal setting, one could first induce the base case and use it to derive $member(a, [a])$. However, this requires an incremental approach and a computationally more expensive use of the background theory (such as for instance the eager or adaptive method of Shapiro's MIS).

⁴Our definition of $flat_int$ is similar to informal notions employed by Cohen (93b) and by many other practitioners of inductive logic programming.

Example 4 This example flattens the clause and interpretation from the previous example. Let $c = \text{sort}([A \mid B], S) \leftarrow \text{partition}(A, B, C, D), \text{sort}(C, E), \text{sort}(D, F), \text{append}(E, [B, F], S)$, and $I = \{\text{append}([1], [2], [1, 2]); \text{partition}(2, [1], [1], []); \text{sort}([], []); \text{sort}([1], [1]); \text{sort}([2, 1], [1, 2])\}$. Then $\text{flat_clause}(c) = \text{sort}(X, S) \leftarrow \text{cons}_p(A, B, X), \text{partition}(A, B, C, D), \text{sort}(C, E), \text{sort}(D, F), \text{cons}_p(B, F, Y), \text{append}(E, Y, S)$, and $\text{flat_int}(I) = \{\text{append}(c_{[1]}, c_{[2]}, c_{[1,2]}), \text{partition}(c_2, c_{[1]}, c_{[1]}, c_{[]}), \text{sort}(c_{[]}, c_{[]}), \text{sort}(c_{[1]}, c_{[1]}), \text{sort}(c_{[2,1]}, c_{[1,2]}), \text{cons}_p(c_1, c_{[2]}, c_{[1,2]}), \text{cons}_p(c_1, c_{[]}, c_{[1]}), \text{cons}_p(c_2, c_{[1]}, c_{[2,1]}), \text{cons}_p(c_2, c_{[]}, c_{[2]})\}$.

An important property of the above flattening mappings is stated by Theorem 2.

Theorem 2 If an interpretation I is a model of a clause C then $\text{flat_int}(I)$ is a model of $\text{flat_clause}(C)$.

Proof: Let us assume that $\text{flat_int}(I)$ is not a model of $\text{flat_clause}(C)$. Then there must exist a substitution θ such that $\text{flat_clause}(C)\theta$ is ground and false in $\text{flat_int}(I)$, i.e. $\text{body}(\text{flat_clause}(C)\theta) \subset \text{flat_int}(I)$ and $\text{head}(\text{flat_clause}(C)\theta) \cap \text{flat_int}(I) = \emptyset$. This implies that $\text{body}(C)\text{unflat_sub}(\theta) \subset I$ and $\text{head}(C)\text{unflat_sub}(\theta) \cap I = \emptyset$, because there exists a one-to-one mapping from atoms $p(c_{t_1}, \dots, c_{t_n}) \in \text{flat_int}(I)$ to atoms $p(t_1, \dots, t_n) \in I$ for regular predicates p (i.e. predicates that are not flattened functors). But this means $C(\text{unflat_sub}(\theta))$ is false in I , which is a contradiction. \square

The theorem does not hold in the reverse direction for clauses which have only infinite non-trivial models.

Example 5 Consider $c = \text{nat}(s(X)) \leftarrow \text{nat}(X)$. Although, $I = \{\text{nat}(s(0)), \text{nat}(0)\}$ is not a model of c , it is easily verified that $\text{flat_int}(I) = \{\text{nat}(c_{s(0)}), \text{nat}(c_0), s_p(c_0, c_{s(0)})\}$ is a model of $\text{flat_clause}(c) = \text{nat}(Y) \leftarrow s_p(X, Y), \text{nat}(X)$.

This example brings us to the heart of the problem and shows how it can be handled. While the original clause/theory has no non-trivial finite models, the flattened theory does. Furthermore, these are finite subsets of the intended interpretation, i.e. finite approximations of the infinite model. This reveals that the flattening approach allows to approximate infinite models of the target theory by finite models of the flattened target theory. To enable the learning of theories with infinite models, one should replace the notion of a model with the notion of an *approximate model*.

Definition 11 An interpretation I is an *approximate model* of a theory T if the flattened interpretation $\text{flat_int}(I)$ is a model of the flattened theory $\{\text{flat_clause}(c) \mid c \in T\}$.

A learnability framework for learning theories with (potentially) infinite models can be obtained by modifying the framework defined in Section 3 as follows: a positive example for a theory is an approximate model of that theory; and the procedure MODEL is replaced by a procedure APPROXIMATE-MODEL, which when called returns an approximate model of the target theory.

The class of jk -CT is then PAC-learnable within this modified framework by the algorithm **Learn-Approximate-Theory** in Figure 2. This algorithm uses the flattened clauses and interpretations for learning, but returns the result in unflattened form.

Learn-Approximate-Theory($j, k, P, F, \epsilon, \delta$)

1. Construct a set V that contains $j \times k$ different variables
 2. $T_0 := \{C \mid C \text{ is range-restricted and } C = h_1 \vee \dots \vee h_r \leftarrow g_1 \wedge \dots \wedge g_s, \text{ where } r + s < k + 1 \text{ and all of the } h_i \text{ and } g_i \text{ are atoms of size at most } j, \text{ constructed of symbols from } P, F \text{ and } V\}$.
 3. $T := \{\text{flat_clause}(c) \mid c \in T_0\}$
 4. **for** $i := 1$ **to** $\max\left(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{8 \times |T_0|}{\epsilon} \log \frac{13}{\epsilon}\right)$ **do**
 - $M := \text{flat_int}(\text{APPROXIMATE-MODEL})$
 - **foreach** $C \in T$ **do**
 - if** C is false in M **then** delete C from T
 5. **return** $\{\text{unflat_clause}(c) \mid c \in T\}$
-

Figure 2: An algorithm for learning jk -CT in the approximate model semantics.

Theorem 3 *The class of jk -CT is polynomial-sample polynomial-time PAC-learnable with one-sided error from positive examples only in the approximate model semantics.*

Proof: The proof is analogous to that of Theorem 1. □

We wish to stress that the use of approximate models is not specific to our approach. Indeed, also in the normal inductive logic programming setting, one needs to approximate infinite models or computations by h -easy approximations, cf. Shapiro’s MIS (83) and Mugleton and Feng’s (90) GOLEM. We believe such approximations are very similar in spirit to ours but their formalization is different.

6 Related Work and Conclusions

To compare our results with PAC-learnability results for the normal ILP setting, let us begin by noting that all of the latter are concerned with learning the definition of a single predicate, i.e. learning a set of definite clauses for a single predicate from a set of true and false facts for that predicate. Several PAC-learnability results exist in the normal ILP setting concerning function-free clauses of bounded length. Džeroski et al. (93) show that k -literal (using explicit unification) non-recursive function-free constrained (where all variables in the body of a clause have to appear in the head) predicate definitions (with arbitrarily many clauses) are polynomially PAC-learnable. The results by Cohen (93b) are closest to ours. Cohen proves that an arbitrary number of function-free non-recursive clauses with at most k literals (again using explicit unification) per clause are PAC-learnable from negative examples only. These can also contain free variables, which appear in the body, but not in the head of a clause. However, adding recursion to the above class of programs makes

learning intractable in the normal ILP setting. Namely, Cohen (93b) also proves that an arbitrary number of function-free, possibly recursive, clauses with at most k literals per clause are not polynomially predictable, and hence are not PAC-learnable, under cryptographic assumptions. All of the above results use a bound j on the arity of background predicates, which is similar to our bound j on the size of atoms. However, they do not impose a bound on the arity of the predicates to be learned, i.e. the predicates in the head of the clauses. Therefore the two results are not really comparable.

Let us however emphasize that several issues, which are known to be hard or which have as yet not been studied in the normal ILP setting, are easy in the nonmonotonic setting. This includes learning clauses with more than one literal in the head, learning clauses with different predicates in the head (i.e. multiple predicate learning), dealing with recursion and even mutual recursion, and using functors. This provides hope that the nonmonotonic setting will allow stronger PAC-learning results than the normal one.

It is not really surprising that these issues are easily dealt with in the nonmonotonic setting. Indeed, the main reasons for this were already pointed out by (De Raedt and Bruynooghe 93, Muggleton and De Raedt 94, De Raedt and Lavrač 93). These include:

- The fact that H is a property of the examples (the examples are models for H) is a stronger condition than $B \wedge H \models P$ and $B \wedge N \wedge H \not\models \square$ (see Theorem 1 of Muggleton and De Raedt 94);
- Clauses in a theory can be considered independently in the nonmonotonic setting. Namely, if a clause c_1 is a property of the examples and a clause c_2 is a property of the examples, then the conjunction $c_1 \wedge c_2$ is also a property of the examples. This does not hold in the normal setting (when learning multiple predicates or recursive definitions), implying a need for backtracking.

The second property is particularly important. Indeed, if the hypothesis space consists of all subsets of clauses in L , then the normal setting has to consider 2^L , whereas the nonmonotonic setting only needs to search L .

Finally, the framework described above is implemented in the CLAUDIEN system (De Raedt and Bruynooghe 1993). The CLAUDIEN system does not, of course, generate all clauses in jk -CT, but searches them using an optimal refinement operator (where all clauses will be generated at most once). Furthermore, CLAUDIEN uses an advanced declarative bias specification mechanism, and employs theorem proving to eliminate logically redundant clauses.

Acknowledgements

The authors of this paper are listed in alphabetic order. Luc De Raedt is supported by the Belgian National Fund for Scientific Research. This paper was written while Sašo Džeroski was visiting the Department of Computer Science, Katholieke Universiteit Leuven, supported by the Commission of the European Communities through PECO Fellowship CIPA3510920370, and PECO Scientific Network on Inductive Logic Programming CIPA3510OCT920044. This research is also part of the ESPRIT Basic Research Project

No. 6020 on Inductive Logic Programming, co-financed by the Flemish Government under contract no. 93/014.

The authors are grateful to the partners in the ESPRIT ILP project, especially to Maurice Bruynooghe, Marc Denecker, Danny De Schreye, and the referees for helpful comments on this work.

References

- [1] H. Adé, L. De Raedt, and M. Bruynooghe. Declarative bias for bottom-up ILP systems. Submitted to Machine Learning, 1994.
- [2] F. Bergadano. Towards an inductive logic programming language. Technical Report Deliverable TO1, ESPRIT Project No. 6020 ILP, Computer Science Department, University of Torino, 1993.
- [3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.
- [4] W. W. Cohen. Pac-learning a restricted class of recursive logic programs. In *Proc. Tenth National Conference on Artificial Intelligence*, pages 86–92. MIT Press, Cambridge, MA, 1993a.
- [5] W.W. Cohen. Learnability of restricted logic programs. In S. Muggleton, editor, *Proc. Third International Workshop on Inductive Logic Programming, ILP'93*, pages 41–71. Jožef Stefan Institute, Ljubljana, Slovenia, 1993b.
- [6] W.W. Cohen. Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, 1994. To appear.
- [7] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proc. Thirteenth International Joint Conference on Artificial Intelligence*, pages 1058–1063, San Mateo, CA, 1993. Morgan Kaufmann.
- [8] L. De Raedt and N. Lavrač. The many faces of inductive logic programming. In *Proc. Seventh International Symposium on Methodologies for Intelligent Systems*, pages 435–449, Berlin, 1993. Springer.
- [9] S. Džeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proc. Fifth ACM Workshop on Computational Learning Theory*, pages 128–135, New York, 1992. ACM Press.
- [10] S. Džeroski, S. Muggleton, and S. Russell. Learnability of constrained logic programs. In *Proc. Sixth European Conference on Machine Learning*, pages 342–347, Berlin, 1993. Springer.
- [11] P.A. Flach. A framework for inductive logic programming. In S.H. Muggleton, editor, *Inductive Logic Programming*, pages 193–211, London, 1992. Academic Press.

- [12] N. Helft. Induction as nonmonotonic inference. In *Proc. First International Conference on Principles of Knowledge Representation and Reasoning*, pages 149–156, San Mateo, CA, 1989. Morgan Kaufmann.
- [13] J.U. Kietz. Some lower bounds for the computational complexity of inductive logic programming. In *Proc. Sixth European Conference on Machine Learning*, pages 115–123, Berlin, 1993. Springer.
- [14] J.U. Kietz and S. Džeroski. Inductive logic programming and learnability. *SIGART Bulletin, Special Issue on Inductive Logic Programming*, 5(1):22–32, 1994.
- [15] J.U. Kietz and S. Wrobel. Controlling the complexity of learning through syntactic and task-oriented models. In S.H. Muggleton, editor, *Inductive Logic Programming*, pages 107–126, London, 1992. Academic Press.
- [16] J.W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 2nd edition, 1987.
- [17] S. H. Muggleton, editor. *Inductive Logic Programming*. Academic Press, London, 1992.
- [18] S.H. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 1994. To appear.
- [19] S.H. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. First Conference on Algorithmic Learning Theory*, pages 368–381, Tokyo, 1990. Ohmsha.
- [20] B.K. Natarajan. *Machine Learning: A Theoretical Approach*. Morgan Kaufmann, San Mateo, CA, 1991.
- [21] C.D. Page. Anti-unification in constraint logics: foundations and applications to learnability in first-order logic, to speed-up learning, and to deduction. Ph.D. thesis. Dept. of Computer Science, University of Illinois at Urbana-Champaign. Report no. UICDCS-R-93-1820, 1993.
- [22] C.D. Page and A.M. Frisch. Generalization and learnability: a case study of constrained atoms. In Muggleton, S. (Ed.), *Inductive Logic Programming*. Academic Press, London, 1992.
- [23] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.
- [24] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In Muggleton, S. (Ed.), *Inductive Logic Programming*. Academic Press, London, 1992.
- [25] C. Rouveirol. Flattening and saturation: two representation changes for generalization. *Machine Learning*, 14:219–232, 1994.
- [26] E.Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.
- [27] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.