Jörg-Uwe Kietz
German National Research Center for Computer Science,
I3-KI, SchloßBirlinghoven, D-5205 St.Augustin, Germany
kietz@gmd.de

Sašo Džeroski
Institut Jožef Stefan,
Jamova 39, 61111 Ljubljana, Slovenia
Saso.Dzeroski@ijs.si

Abstract

The paper gives an overview of theoretical results in the rapidly growing field of inductive logic programming (ILP). The ILP learning situation (generality model, background knowledge, examples, hypotheses) is formally characterized and various restrictions of it are discussed in the light of their impact on learnability. The two dominant models of learnability, PAC-learning and identification in the limit, are extended to take into account the ILP learning situation. Several learnability results for logic programs are then presented, both positive and negative.

1 Introduction

Most successes within the field of Machine Learning have been achieved with systems learning in propositional logic. The theory of learnability, e.g., PAC-learnability, has been consequently mostly concerned with propositional logic. But, despite their successes, propositional learning approaches suffer from the limited expressiveness of their hypothesis language and the inability to use background knowledge. The field of Inductive Logic Programming (ILP), concerned with the induction of first-order Horn clauses[1] from examples and background knowledge, has recently received increased attention [39, 29, 8, 24, 28]. The problem addressed by Inductive Logic Programming can be described formally as follows:

Definition 1 (ILP Learning Problem)
Given:

- *a correct consequence relation* $\vdash$ *for a first-order language* L, *i.e., for all* $A, B \in L$: *if* $A \vdash B$, *then* $A \models B$,
- *background knowledge* B *in language* LB: $B \in LB \subseteq L$
- *positive and negative examples* $E = E^+ \cup E^-$ *in language* $LE \subseteq L$ *consistent with* B, $(B, E \not\vdash \square)$ *and not a consequence of* B $(\forall e \in E : B \not\vdash e)$, *and*
- *hypothesis language* $LH \subseteq L$.

Find a hypothesis $H \in LH$ *such that:*

(I) $(B, H, E \not\vdash \square)$, *i.e.,* H *is consistent with* B *and* E,

(II) $(B, H \vdash E^+)$, *i.e.,* $H \wedge B$ *explain* E^+, *and*

(III) $(\forall e \in E^- : B, H \not\vdash e)$, *i.e.,* $H \wedge B$ *do not explain* E^-.

The tuple $(\vdash, LB, LE, LH)$ *is called the* ILP-learning problem. *Deciding whether there exists such an* $H \in LH$, *is called the* ILP-consistency problem. *An algorithm which accepts any* $B \in LB$ *and* $E \subseteq LE$ *as input and computes such an* $H \in LH$ *if it exists, or "no" if it does not exist is called an* ILP-learning algorithm *for* $(\vdash, LB, LE, LH)$. *As* LB, LE *and* LH *are subsets of first-order clauses,* $(\vdash, LB, LE, LH)$ *is also called the ILP-problem and a learning algorithm for* $(\vdash, LB, LE, LH)$ *is called an ILP-algorithm.*

The paper first extends the two dominant models of learnability, PAC-learning and identification in the limit, to be suitable for the ILP learning problem. It then gives a general result connecting the property of PAC-learnability to the polynomial solvability of the ILP-learning problem. Section 3 gives an overview of common restrictions of the ILP-problem, and their impact on the complexity of the ILP-problem. We then prove one positive and two negative results concerning the PAC-learnability of restricted classes of logic programs that are very close together. We conclude with a discussion of our results in the context of related results on the learnability of logic programs.

2 Learnability Models

The ILP problem defined above is concerned with constructing a hypothesis from a given finite set of examples. The theory of learnability is additionally concerned with the sampling process. There are two main approaches to learnability theory as described in this section.

- Gold's [14] identification in the limit. This approach is derived from computability theory, and deals with finite time convergence of a learning procedure on an infinite example stream.

[1] We assume the reader is familiar with logic programming terminology as described in the textbook by Lloyd [26].

Approximately-Correct(PAC) learning. This is derived from cryptography and computational complexity theory and deals with probabilistic identification from finite (polynomial) sets of examples.

2.1 Identification in the Limit

Identification in the limit models to some extent the Popperian view [38] of the scientific discovery process. As it it impossible to ensure the correctness of a theory with infinitely many consequences from only finitely many examples, the best we can do is to build hypotheses which explain the current examples and always look for new examples which can improve our theory. This infinite process of gathering information and correcting theories may converge to the correct theory. The theory of identification in the limit is concerned with finding hypothesis spaces and algorithms where this convergence can be guaranteed. Let us define this more formally in the context of ILP.

Definition 2 (Identification in the limit) *Let $E_\infty = \{e_1, e_2, e_3, \ldots\}$ denote an infinite stream of examples, where $e_i \in LE \times \{+, -\}$, and let E_∞ be a complete enumeration of the examples in LE, i.e., $\forall e \in LE : \exists i : (e_i = \langle e, + \rangle \vee e_i = \langle e, - \rangle)$. Let $E_i = \{e_1, e_2, e_3, \ldots, e_i\}$ denote the finite set of examples known at time i. Let $E_i^+ = \{e \mid \langle e, + \rangle \in E_i\}$ be the set of all positive examples known at time i and $E_i^- = \{e \mid \langle e, - \rangle \in E_i\}$ be the set of all negative examples known at time i. Also, let us say that background knowledge B is suitable for E_∞ and LH iff $\exists H \in LH : (B, H \vdash E_\infty^+ \wedge \forall e \in E_\infty^- : B, H \nvdash e)$.*

A learning problem $(\vdash, LB, LE, LH)$ is identified in the limit by an algorithm ILEARN iff for every sample E_∞ and suitable background knowledge $B \in LB$ ILEARN accepts E_∞ incrementally as input, computes after every example an $H_i \in LH$ and there exists time i, such that $\forall j \geq i : H_i = H_j$ and $(B, H_i \vdash E_\infty^+ \wedge \forall e \in E_\infty^- : B, H_i \nvdash e)$.

An algorithm ILEARN does consistent identification in the limit if, in addition to the requirements in the above definition, the following property holds:
$\forall i : (B, H_i \vdash E_i^+ \wedge \forall e \in E_i^- : B, H_i \nvdash e)$.

The main identification in limit result in ILP is due to Shapiro [43]. He showed that his MIS system can identify in the limit any ground atomic complete h-easy Horn clause theory from ground atoms as examples. Roughly speaking, a ground atomic complete h-easy Horn clause theory is a set of Horn clauses, for which all ground atoms true in the intended model can be inferred in h steps, where h is a total recursive function from ground atoms to natural numbers. Shapiro also shows that this is the most powerful result obtainable for any consistent identification in the limit learning algorithm.

quired for the ILP problem) are not strongly connected. They are somewhat orthogonal features of learning. On the one hand, a sequence of hypotheses $H_1, H_2, H_3, \ldots$ produced by an ILP-algorithm from the example sets $E_1, E_2, E_3, \ldots$ need not necessarily identify H in the limit. Suppose $LE \subseteq LH$. $H_i = E_i^+$ is then always a consistent solution to the corresponding ILP-problem. However, there need not exist time i, such that $\forall j \geq i : H_j = H_i$. Obviously, this is the case if positive examples keep arriving, i.e., $\neg \exists i : E_\infty^+ = E_i^+$. Therefore, one often requires that $LE \not\subseteq LH$, or restricts the introduction of disjunctions. However, even in this case, computing consistent hypotheses is not enough to ensure identification in the limit.

On the other hand, identification in the limit does not in general require that every H_i is consistent with the corresponding E_i. It has been proved by Wiehagen and Zeugmann [45] that this consistency requirement actually prohibits identification in the limit for some classes of recursive functions that are identifiable in the limit without this consistency requirement.

2.2 PAC-learning

Valiant's [44] Probably Approximately Correct (PAC) learning framework relaxes the requirement of identification in the limit in that the learning algorithm need only produce a hypothesis that with high probability is a good approximation of the target concept. To make precise the notions of "high probability" and "good approximation", the examples are produced by an unknown, but fixed probability distribution over the example space. On the other hand, Valiant's framework limits the resources that a learning algorithm is to use. The PAC-learning framework requires that learning finishes after a polynomial amount of time and accordingly takes at most polynomially many examples for learning. As PAC-learning was originally defined in the context of learning Boolean functions, "polynomial" originally meant polynomial in the number of Boolean variables used for the example description, the probability of success and the quality of the approximation required.

As ILP operates not only on examples, but also on background knowledge, we need a more sophisticated measure of complexity. The difficulty of learning, and even the possibility to learn, depends on the background knowledge. If we have no background knowledge and are not capable of introducing new predicates, the concept `reverse(List,ReversedList)` is not expressible at all. If we have a definition of `nrev(DoList,DoneList,Reversed)` in the background knowledge it is easy to define $reverse(L, RL)$ as $reverse(L, RL) \leftarrow nrev(L, [], RL)$. If we have `append(List1,List2,AppendedLists)` in the background knowledge `reverse` is learnable, if LH contains

$reverse([],[]).$
$reverse([F|R],L) \leftarrow$
$reverse(R,RL), append(RL,[F],L).$

These examples show that the complexity (coding size) of a concept depends on the hypothesis language as well as on the available background knowledge. We write LH(B) for the hypothesis language using the background knowledge B. In the examples, LH(B) contains clauses using `nrev` or `append`, respectively. Let n_e, n_b, n_c denote size complexity measures for the examples in LE, the background knowledge in LB, and the concepts in the language LH(B). Polynomial PAC-learnability for the ILP-setting can then be defined as follows.

Definition 3 (Polynomial PAC-learning) *A learning problem $(\vdash, LB, LE, LH)$ is polynomially PAC-learnable, iff there is an algorithm PLEARN and a polynomial function $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_c, n_b)$, so that for every $n_c > 0$, every $n_e > 0$, every $n_b > 0$, every background knowledge $B \in LB$ of complexity n_b or less, every concept $C \in LH(B)$ with complexity n_c or less, every $\epsilon : 0 < \epsilon < 1$, every $\delta : 0 < \delta < 1$, and every probability distribution D, for any set of examples $E = E^+ \cup E^-$ of C, drawn from LE according to D and containing at least $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_c, n_b)$ examples*

(I) PLEARN, on input E, B, ϵ, and δ outputs a hypothesis $H \in LH$ such that

$$P(D(Cover(H)\Delta Cover(C)) > \epsilon) < \delta,$$

where $Cover(X) = \{e \in LE | X, B \vdash e\}$ and Δ denotes symmetric set difference,

(II) PLEARN runs in time polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_c, n_b$ and the number of examples, and

(III) For all $e \in LE$, the truth of the statement $e \in Cover(H)$ can be tested in polynomial time.

The polynomial function $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_c, n_b)$ is called the sample complexity of the algorithm PLEARN.

If all of the conditions of the above definition are satisfied, except for $H \in LH$, we say that the learning problem is polynomially predictable.

Polynomial PAC-learnability as defined above and the solvability of the ILP-problem in random polynomial time are strongly related. An ILP-algorithm is always a good candidate to solve a polynomial PAC-learning problem, if it runs in time polynomial in n_e, n_c, n_b. Given a particular example set, no learning algorithm can guarantee a better hypothesis with higher probability than an algorithm which computes a hypothesis consistent with all the given examples. However, polynomial solvability of the ILP-problem is not enough to also show that a good hypothesis is always found from only polynomially many examples.

But what about the other direction? Does polynomial PAC-learnability ensure polynomial solvability of the ILP-problem? We use an adaptation of a theorem of Pitt and Valiant, also proved as theorem 6.2.1 by Anthony and Biggs [1] to show how a polynomial PAC-learner PLEARN can be used to solve the ILP-problem in random-polynomial time.

Theorem 1 *If a learning problem $(\vdash, LB, LE, LH)$ is polynomially PAC-learnable by an algorithm PLEARN, then the ILP-problem for $(\vdash, LB, LE, LH)$ is in RP, or turning it the other way around, if the ILP-problem is not in RP, then $(\vdash, LB, LE, LH)$ is not polynomially PAC-learnable.*

Proof: Suppose that $E = E^+ \cup E^- \subseteq LE$ is an example set of size s for a concept $C \in LH$. Let D be a probability distribution defined as:

$$D(e) = \begin{cases} 1/s, & \text{if } e \in E \\ 0, & \text{otherwise} \end{cases}$$

Let PLEARN be the polynomial PAC-learner for $(\vdash, LB, LE, LH)$. Set $\delta = 1/2$, $\epsilon = 1/s$, and run PLEARN on $m(\frac{1}{\epsilon}, \frac{1}{\delta}, n_e, n_c, n_b)$ examples drawn from LE according to D, i.e, drawn from E by a random number generator. The PAC-learnability property of PLEARN ensures that the probability that the error of the output H is less than $1/s$ is greater than $1 - 1/2 = 1/2$. Since there are no examples with probability less than $1/s$, if H has error less than $1/s$, it has error 0, i.e., it is consistent with all examples in E. The probability that H is a solution to the ILP-problem is thus $1/2$.

As we can verify that H is consistent with the examples in polynomial time (due to the PAC-learnability property (III)) and PLEARN runs on a polynomial sample in polynomial time, we get a correct solution to the ILP-problem with probability $1/2$ in polynomial time, i.e., we have constructed a random-polynomial time ILP-algorithm from PLEARN.□

3 Complexity Problems in ILP

The general ILP problem ($\models$, Horn clauses, ground atoms, Horn clauses) is undecidable. It is clear that undecidable ILP-problems are neither consistently identifiable in the limit, nor PAC-learnable. Therefore, a main topic in ILP research is the identification of subclasses of first-order logic that are efficiently learnable. As the examples are already restricted to ground atoms, possible choices to restrict the ILP-problem are the consequence relation (also called the generalization model) the background knowledge and the hypothesis language. In this section we will review possible restrictions along these dimensions.

One parameter of the ILP-problem is the consequence relation. Two main relations have been used in the ILP framework:

- implication, used in [37, 43, 30, 20, 19], and
- subsumption, used in [35, 36, 31, 39, 21].

3.1.1 Implication

Implication has one main advantage over subsumption. It is not only a correct, but also a complete consequence relation between clauses. Due to a proof of decidability between Horn clauses by Niblett [32] (which is correct only for a limited case as we will show later) this advantage seems to make implication superior to subsumption. However, as newer results in logic programming show, implication is undecidable even for some very limited cases. Here we only recall three results and indicate the immediate consequences for the decidability of the ILP-problem.

Theorem 2 (Marcinkowski and Pacholski [27])
The satisfiability of a Horn clause program, consisting of a set of ground facts, one ground query, and one Horn clause with three literals is undecidable in the general case.

Corollary 3 *The ILP-problem ($\models$, ground facts, ground facts, a 3-literal Horn clause) is undecidable in the general case.*

Theorem 4 (Schmidt-Schauss [42])
Satisfiability of a Horn clause program, consisting of one ground fact, one ground query, and two Horn clauses with two literals is undecidable in the general case.

Corollary 5 *The ILP-problem ($\models$, a ground fact, ground facts, two 2-literal Horn clauses) is undecidable in the general case.*

Theorem 6 (Hanschke and Würtz [16])
Satisfiability of a Horn clause program, consisting of one nonground fact, one nonground query, and one Horn clause with two literals is undecidable in the general case.

Corollary 7 *The ILP-problem ($\models$, a nonground fact, nonground facts, a 2-literal Horn clause) is undecidable in the general case.*

The result in theorem 2 contradicts the termination proof for algorithm 6.1 in Niblett [32]. Indeed, there exist counterexamples to the central assumption of Niblett's proof, namely that a proof will only contain terms less complex than the terms in the two Horn clauses[2].

[2]This was discovered in a discussion of theorem 2 between Kietz and Niblett

$$\begin{aligned} D = i(X4, Y1, X5, Y2) &\leftarrow i(X1, X2, X3, f(X4)), \\ &\quad\ i(f(X5), X6, X7, X8). \\ C = i(c, c, c, c) &\leftarrow i(f(a), f(a), f(a), f(a)). \end{aligned}$$

In any proof of $D, C_{body} \vdash_{SLD} C_{Head}$ the term $f(f(c)))$ which is more complex than any term in C and D must be constructed. $\diamond$

This counterexample is possible as the head of D contains variables not in the body of D. These variables can absorb the more complex terms needed in intermediate states of the proof. If we limit the Horn clause D to be generative, then Niblett's assumption becomes valid.

Definition 4 (Generative Hornclause)
A Horn clause is generative if every variable in the head also occurs in the body of the clause.

Theorem 8 (Correction of Niblett [32])
Let D, C be two Horn clauses, where D is generative, then $D \models C$ is decidable, or equivalently, let F be a set of ground unit clauses, D be a generative Horn clause, and Q a ground query. It is decidable whether Q can be inferred from F and D by SLD-resolution ($F, D \vdash_{SLD} Q$).

Corollary 9 *The ILP-problem ($\models$, ground facts, ground facts, a generative Horn clause) is decidable.*

This section shows that the use of implication in the ILP-problem, suggested recently by Muggleton [30] and Idestam-Almquist [20, 19], can be a very dangerous choice. Namely, the corresponding ILP-problem is undecidable, even for very limited cases.

3.1.2 Subsumption

An alternative to implication is θ-subsumption, first defined by Robinson [41] and used in the ILP-framework by Plotkin [35].

Definition 5 (θ-subsumption)
A clause D θ-subsumes a clause C ($D \vdash_\theta C$), iff there exists a substitution θ such that $D\theta \subseteq C$.

θ-subsumption is a correct consequence relation, i.e., if $D \vdash_\theta C$ then $D \models C$. However, as θ-subsumption between clauses is an NP-complete problem [13], i.e., decidable, it must be incomplete with respect to implication. Gottlob [15] has shown that this incompleteness can be precisely characterized.

Theorem 10 (Gottlob [15])
If D is not self-resolving and C is not tautological, then $D \models C$ is equivalent to $D \vdash_\theta C$

cases of implication are NP-hard and there is an immediate corollary concerning the corresponding ILP-problem.

Corollary 11 *The ILP-problem ($\vdash_\theta$, {}, ground Horn clauses, a Horn clause) is NP-hard. Therefore, it is not polynomially PAC-learnable.*

The use of θ-subsumption as the generalization model has another drawback. It excludes the use of background knowledge, as θ-subsumption is only defined between two Horn clauses. To repair this deficiency Plotkin [36] has defined the notion of subsumption relative to a background theory. Buntine [3] defined a special case of relative subsumption called generalized subsumption. As this paper is mostly concerned with subsumption between Horn clauses with the same head, we will use Buntine's definition.

Definition 6 (Generalized subsumption)
Let $B \subseteq LB$ be background knowledge and let σ be a Skolem substitution for a Horn clause C. A clause D is more general than a clause C w.r.t. generalized subsumption, $D \vdash_B C$, iff there is a substitution θ, such that $D_{Head}\theta = C_{Head}$ and $B, C_{Body}\sigma \models C_{Body}\theta\sigma$.

Buntine has shown that θ-subsumption is a special case of generalized subsumption occurring when the background knowledge is empty, i.e., we have an extension of θ-subsumption incorporating background knowledge. However, the background knowledge used in ILP programs must always be restricted, otherwise the ILP problem inherits the undecidability of the deduction process within the background knowledge.

3.2 Common Restrictions of LB

A common restriction in ILP programs (e.g., [31, 39]) is the restriction to ground background knowledge and ground unit clauses as examples.

Definition 7 (Ground background knowledge)
The background knowledge B is ground if it consists of ground unit clauses only. A clause is ground if it does not contain any variables.

The restriction to ground unit clauses as examples and ground background knowledge enables us to use $\theta - subsumption$ as an inference procedure completely incorporating background knowledge in the following way.

Lemma 1 *The ILP problem ($\vdash_B$, ground B, ground unit E,LH) is equivalent to the ILP problem ($\vdash_\theta$, {},E_{new},LH), where E_{new} is defined as $E^+_{new} := \{e \leftarrow B \mid e \in E^+\}$, and $E^-_{new} := \{e \leftarrow B \mid e \in E^-\}$.*

An immediate corollary from lemma 1 and corollary 11 is:

unit E, a Hornclause) is NP-hard.

One way to avoid the restriction to ground background knowledge is the use of generative Horn clauses and a depth-bounded inference process to generate finite ground background knowledge prior to learning (e.g., [31, 22]).

Another common restriction applied to LB is the restriction to Datalog clauses (e.g., [39, 22]). A similar restriction is applied to LH, namely, the restriction to function-free clauses.

Definition 8 (Datalog clause)
A Horn clause is called Datalog clause if all of the terms appearing in it are either variables or constants (function symbols of arity 0).

Definition 9 (Function-free clause)
A clause is called function-free if it contains no function symbols, including constants.

Datalog knowledge has a positive effect on deduction. It has been proved that inferring ground background knowledge can be done completely in time polynomial in the size of B if B consists only of generative Datalog clauses [46] and there is a fixed maximum arity of predicates and a fixed maximum number of clause literals in the background knowledge.

Definition 10 (Bounded-arity language)
The language for background knowledge LB is of bounded arity if there exist an integer j, which is greater than the maximum arity of any of the predicates allowed in LB.

3.3 Common Restrictions of LH

The bounded-arity restriction is also used by Page and Frisch [33] to prove the PAC-learnability of a special kind of hypothesis language LH called *constrained clauses*.

Definition 11 (Constrained clause)
A clause is constrained if all variables in the body also occur in the head.

Muggleton and Feng [31] have proposed a restriction of LH to ij-determinate Hornclauses. In contrast to the above restrictions on LB and LH, the determinacy restriction is not a purely syntactic restriction.

Definition 12 (Determinate Clause)
A Hornclause h is determinate with respect to the examples E and the background knowledge B if every term t in h is determinate, i.e., linked by a determinate linking-chain. A term occurring in the head of h is determinate, i.e., is linked by a determinate linking-chain of length 0. Let $h = \{A, \neg B_1, \ldots, \neg B_m, \neg B_{m+1}, \ldots, \neg B_n\}$ be ordered. The term t found in $\neg B_{m+1}$ is linked by a determinate linking-chain of length $i+1$, iff all the terms

by determinate linking-chains of length at most i and for every substitution θ such that $A\theta \in E$ and $B \vdash \{B_1, \ldots, B_m\}\theta$ there is a unique substitution δ, whose domain is the variables in t such that $B \vdash B_{m+1}\theta\delta$. The determinate depth of a term is the minimal length of its determinate linking chains.

If the substitutions for the determinate term t in literal $\neg B_{m+1}$ depend on the substitutions applied to j_d other terms in the same literal, we say that t is of degree j_d. A clause with maximum determinate depth of terms i and maximum degree of terms j_d is called ij_d-determinate. If we have Datalog clauses, an arity bound j on the predicates in the body effectively reduces the maximum degree of any term, i.e., $j_d \leq j - 1$. Thus, if a Datalog clause is determinate with maximum depth of terms i, it is $i, j-1$-determinate, and consequently ij-determinate.

Effectively, the determinacy restriction means that given ground substitutions for the variables in the head of a clause (examples are ground), the ground values for all the variables in the body of the clause are uniquely determined. Muggleton and Feng have used this ij-determinate restriction to prove that the ILP-problem (subsumption, ground background knowledge, ground unit clause examples, one ij-determinate Hornclause) is solvable in polynomial time [31].

Definition 13 (Linked Horn clause)
A Horn clause is linked if all of its literals are linked. A literal is linked if at least one of its terms is linked. A term is linked with linking-chain of length 0 if it occurs in the head of the clause. A term in a literal is linked with a linking-chain of length $d+1$, if another term in the same literal is linked with a linking-chain of length d. The depth of a term is the minimal length of its linking-chains.

A clause which is not determinate, has maximal depth of terms i and maximal arity of literals j, is called ij-nondeterminate.

Clearly, terms in nondeterminate clauses need not have a determinate linking-chain. If a term has a *determinate linking-chain*, this is also a (nondeterminate) *linking-chain*, but not vice versa. Therefore, the *determinate depth* of a term is greater than or equal to the (nondeterminate) *depth* of a term.

Definition 14 (k-literal predicate definition)
A k-literal clause is a program clause with at most k literals in its body. A k-literal predicate definition consists of k-literal clauses with the same predicate in the head.

Definition 15 (k-clause predicate definition)
A k-clause predicate definition consists of up to k program clauses with the same predicate in the head.

literals of the form *not* A can appear in the body, where A is an atom. For these literals, the 'negation as failure' interpretation [4] is assumed. A goal *not* A succeeds (the literal *not* A is true) under the 'negation as failure' interpretation if the goal A finitely fails, i.e., cannot be proved using the resolution rule of inference. Given the above definitions, a k-clause predicate definition corresponds to a k-term first-order DNF formula and a k-literal predicate definition corresponds to a first-order k-DNF formula.

Definition 16 (k-discriminative predicate definition)

Let $h = \{A, \neg B_1, \ldots, \neg B_{m-1}, \neg B_m, \ldots, \neg B_n\}$ be a determinate clause. $\neg B_m$ is called a determinate literal if it contains variables that do not appear in the clause $\{A, \neg B_1, \ldots, \neg B_{m-1}\}$, and discriminative literal otherwise. A k-discriminative clause is a determinate clause with at most k discriminative literals in its body. A k-discriminative predicate definition consists of k-discriminative clauses with the same predicate in the head.

The intuition behind the above definition is that discriminative literals can distinguish between positive and negative examples while determinate literals cannot.

Example 2
Let us illustrate the above definitions on a simple ILP problem. The task is to define the target predicate $grandmother(X, Y)$, which states that person X is the grandmother of person Y, in terms of the predicates $father$ and $mother$. The training examples and background knowledge are given in table 1. They consist of ground unit Datalog clauses.

A definition of the target predicate in terms of background knowledge predicates is

$$\begin{array}{c} \forall X \forall Y : grandmother(X, Y) \leftrightarrow \\ [\exists Z : father(Z, Y) \wedge mother(X, Z)] \vee \\ [\exists U : mother(U, Y) \wedge mother(X, U)] \end{array}$$

or in logic programming notation

$$\begin{array}{l} grandmother(X, Y) \leftarrow \\ \quad father(Z, Y), mother(X, Z). \\ grandmother(X, Y) \leftarrow \\ \quad mother(U, Y), mother(X, U). \end{array}$$

In the above terminology, this hypothesis is *determinate* (but not *constrained*), because the variables Z in $father(Z, Y)$ and U in $mother(U, Y)$ are determinate, i.e., have only one possible value given a ground substitution for X and Y. The hypothesis is function-free, the maximum depth of variables is 1, and the maximum arity of background knowledge predicates is 2. The hypothesis is thus 12-determinate. As it consists of two

$grandmother(ann, bob).\oplus$	$father(zak, tom).$	$father(pat, ann).$	$father(zak, jim).$
$grandmother(ann, sue).\oplus$	$mother(ann, tom).$	$mother(liz, ann).$	$mother(ann, jim).$
$grandmother(bob, sue).\ominus$	$father(tom, sue).$	$father(tom, bob).$	$father(jim, dave).$
$grandmother(tom, bob).\ominus$	$mother(eve, sue).$	$mother(eve, bob).$	$mother(jean, dave).$

Table 1: A Simple ILP Problem

clauses, each of them with two literals in the body, the hypothesis is a 2-clause and a 2-literal predicate definition. Also, as one of the literals in each of the clause bodies is determinate and the other is discriminative, the hypothesis is 1-discriminative.

However, the logically equivalent hypothesis

$grandmother(X, Y) \leftarrow$
$\quad mother(X, Z), father(Z, Y).$
$grandmother(X, Y) \leftarrow$
$\quad mother(X, U), mother(U, Y).$

is not determinate, since the variable Z in the literal $mother(X, Z)$ is not determinate. Namely, for $\theta = \{X/ann, Y/bob\}$, there are two substitutions ($\delta = \{Z/tom\}$ and $\delta = \{Z/jim\}$), such that $mother(X, Z)\theta\delta \in B$. In the above terminology, this hypothesis is 12-nondeterminate. $\diamond$

4 Positive Result

Our positive learnability result is based on the LINUS [23] approach of transforming an ILP problem to propositional form, then using learnability results for the propositional case. Džeroski and Lavrač [10] have shown that the ILP-problem of learning constrained function-free clauses can be transformed into a polynomially larger propositional representation. The features in the latter correspond to all possible applications of the background predicates to the arguments of the target predicate. We extend these results to determinate clauses. We first present the transformation algorithm and then illustrate its operation on the example from table 1.

To transform the ILP problem of constructing a definition for target predicate $q(X_1, X_2, ..., X_n)$ proceed as follows. First, construct a list F of all literals that use the predicates $p_1, p_2, ..., p_l$ from the background knowledge and contain determinate variables of depth at most i. The literals that introduce new variables are excluded from this list, as they do not distinguish between positive and negative examples (the new variables have a unique value for any example, due to the determinacy restriction). The resulting list is the list of features used for propositional learning. Next, transform the examples to propositional form. For each example, the truth value of each of the propositional features is determined by calls to the background knowledge base. This is done by algorithm 1.

Algorithm 1

1. $V_0 = \{X_1, ..., X_n\}$
2. $L = \{\}$
3. *For* $r = 1$ *to* i *do*
   - $D_r := \{p_s(Y_1, ..., Y_{j_s}) \mid p_s \in B,$ $q(X_1, ..., X_n) \leftarrow L, p_s(Y_1, ..., Y_{j_s})$ *is determinate and contains at least one variable that is not in* $V_{r-1}\}$.
   - $L := L \cup D_r$.
   - $V_r = V_{r-1} \cup \{Y | Y$ *appears in a literal from* $D_r\}$.
4. $F = \{p_s(Y_1, ..., Y_{j_s}) |\ p_s \in B,\ Y_1, ..., Y_{j_s} \in V_i\} - L$.
5. *For each* $e = q(a_1, ..., a_n) \in E$ *do*
   - *Determine the ground substitutions for (values of) the variables in* V_i *by executing the body of the clause* $q(X_1, ..., X_n)\theta \leftarrow L\theta$, *where* $\theta = \{X_1/a_1, ..., X_n/a_n\}$.
   - *Given the ground substitutions for the variables in* V_i, *determine* $F(e)$, *the tuple of truth values of the literals in* F, *by querying the background knowledge* B.
   - $F(e)$ *is an example of a propositional concept c (positive if e is a positive or negative if e is a negative example).*

Two types of queries have to be posed to the background knowledge base. The first type are *existential* queries, which are used to determine the values of the new variables. Given a partially instantiated goal (literal containing variables that are not bound), an existential query returns the set (possibly empty) of all ground substitutions (bindings) for the unbound variables which make the literal true. For example, the query $mother(ann, A)$, where A is an unbound new variable, would return the set of substitutions $\{\{A/tom\}, \{A/jim\}\}$. The other type of queries are ground (*membership*) queries about background knowledge predicates, where the goal is completely bound. These are used to determine the truth values of the propositional features. For example, the

$g(X,Y)$	X	Y	$f(U,X)$	$f(V,Y)$	$m(W,X)$	$m(Z,Y)$	...	$m(X,V)$	$m(X,Z)$	...
c			U	V	W	Z		x_1	x_2	
1	*ann*	*bob*	*pat*	*tom*	*liz*	*eve*		1	0	
1	*ann*	*sue*	*pat*	*tom*	*liz*	*eve*		1	0	
0	*bob*	*sue*	*tom*	*tom*	*eve*	*eve*		0	0	
0	*tom*	*bob*	*zak*	*tom*	*ann*	*eve*		0	0	

Table 2: Propositional Form of a Simple ILP Problem

query $mother(bob, tom)$ yields the answer $false$ and the corresponding feature takes the value 0.

To illustrate the work of algorithm 1, consider the ILP example from table 1. In this case, we have maximum arity of predicates in B $j = 2$, maximum determinate depth of H $i = 2$, number of predicates in B $l = 2$, and arity of the examples $n = 2$. A literal $father(X, A)$, where A is a new variable, is not determinate, as a man can have several children. However, if instead A is old and X is new, the literal is determinate, since each person has exactly one father. As the target predicate is $grandmother(X, Y)$, we have $V_0 = \{X, Y\}$ and $D_1 = \{f(U, X), f(V, Y), m(W, X), m(Z, Y)\}$, where f and m stand for $father$ and $mother$, respectively.

This gives $L_1 = D_1$, $V_1 = \{X, Y, U, V, W, Z\}$. F includes literals such as $f(X, X), f(X, Y), f(Z, Y), f(W, X)$ and similarly $m(Z, Z), m(V, Y), m(W, W), m(U, X)$. In fact, the pairs of arguments of f and m are all the pairs from the Cartesian product $V_1 \times V_1$, excluding the pairs that produce literals from D_1.

To illustrate the transformation process, table 2 gives two features and their propositional values, as well as the values of the variables introduced by the determinate literals, generated for the ILP problem as defined by the training examples and background knowledge from table 1.

The transformation process from a propositional DNF formula to a logic program proceeds as follows. The DNF formula is first rewritten into a set of propositional Horn clauses. For example, suppose a propositional learner induces the concept $c \leftrightarrow x_1 \vee x_2$ from the examples in table 2. This is rewritten into $c \leftarrow x_1$ and $c \leftarrow x_2$. Each propositional feature is than replaced with the corresponding literal from the propositional table. In this way, we get the following two Horn clauses: $g(X, Y) \leftarrow m(X, V)$ and $g(X, Y) \leftarrow m(X, Z)$. Namely, the target predicate $g(X, Y)$ corresponds to the concept c, whereas the literals $m(X, V)$ and $m(X, Z)$ correspond to the features x_1 and x_2. As these literals use the new variables V and Z, we must include the literals that introduced the new variables in the above clauses. For each clause the following process is repeated, until all variables in the body and not in the head are introduced by literals from L:

- Choose a variable A that appears in the body, but not in the head, and is not introduced by a literal from L.
- Add the first literal in L in which A appears, i.e., the literal that defines A, to the body of the clause.

The literals in the body are then ordered so that all literals from L appear first in the same relative order as in L, followed by the literals that correspond to the propositional features. The variables V and Z in our example are introduced by the literals $f(V, Y)$ and $m(Z, Y)$ from L and the final version of the clauses is as follows:

$$g(X, Y) \leftarrow f(V, Y), m(X, V).$$

$$g(X, Y) \leftarrow m(Z, Y), m(X, Z).$$

Let us now state a positive result regarding the PAC-learnability of determinate function-free program clauses.

Lemma 2 *Algorithm 1 transforms the ILP problem defined by a set of m examples E of the target predicate $q(X_1, X_2, ..., X_n)$, background predicates $p_1, p_2, ..., p_l$ of maximum arity j, and maximum depth of variables i, to a propositional form in time*

$$O(n_b m (4jln)^{j^{i+1}}),$$

assuming that the examples and the background knowledge consist of ground facts and n_b is the size of the background knowledge.

Proof: Let $v_r = |V_r|$ in algorithm 1. We have $v_0 = n$ and $v_r \leq v_{r-1} + l \sum_{k=1}^{j-1} k \binom{j}{k} v_{r-1}^{j-k}$, since there are at most $\binom{j}{k} v_{r-1}^{j-k}$ applications of each predicate from the background knowledge introducing exactly k new variables. It can be easily proved that $v_i \leq (4jln)^{j^i}$. Each of the l predicates from the background knowledge can be applied to the variables in V_i in at most v_i^j ways. The number of features in F is thus upper bounded by $n_i = l v_i^j \leq l(4jln)^{j^{i+1}}$.

As the background knowledge consists of ground facts, each query to it can be answered in time proportional

ple to propositional form takes $O(n_b v_i)$ time to determine the values of variables in V_i and $O(n_b n_i)$ time to determine the truth values of features in F, altogether $O(n_b(4jln)^{j^{i+1}})$. For m examples, the transformation process takes $O(n_b m(4jln)^{j^{i+1}})$ time. □

If we allow the background knowledge to be nonground, but still Datalog and generative, the transformation can be still completed in polynomial time. Namely, the nonground background knowledge can be transformed to ground form in time polynomial in the size of the background knowledge [46].

Theorem 13 *ij-determinate k-discriminative nonrecursive function-free predicate definitions are polynomially PAC-learnable under arbitrary distributions.*

Proof: After transforming the problem to propositional form, we use an algorithm for learning k-DNF [44, 17]. When transforming the induced k-DNF formula to logic program form we have to include the necessary determinate literals (that introduce new variables). This may change the length of each of the clauses, but will not change the corresponding number of discriminative literals (that do not introduce new variables). The resulting predicate definition is therefore k-discriminative. The time taken by the transformation to logic program form is $O(h(4jln)^{j^{i+1}})$, where h is the size of the induced k-DNF formula and $(4jln)^{j^{i+1}}$ is an upper bound on the total number of determinate literals. The polynomial learnability under arbitrary distributions of programs consisting of non-recursive function-free clauses with variables of depth up to i and at most k literals that do not introduce new variables then follows from the polynomial learnability of k-DNF under arbitrary distributions. Note that item (III) in the definition of PAC-learnability is satisfied by taking into account the transformation carried out by algorithm 1. □

The above proof is very much in the spirit of prediction-preserving reducibilities [34]. In fact, the transformation carried out by algorithm 1 is a prediction-preserving reduction as it is carried out in polynomial time. This allows us to state that k-clause ij-determinate nonrecursive function-free predicate definitions are polynomially predictable under arbitrary distributions, as k-term DNF is polynomially predictable. Džeroski et al. [11], on the other hand, prove that k-clause ij-determinate definite nonrecursive function-free predicate definitions are polynomially PAC-learnable under "simple" distributions [25].

5 Negative Results

In the previous section, we proved that k-discriminative nonrecursive function-free ij-determinate programs are polynomially PAC-learnable under arbitrary distributions. problem can be reduced to an equivalent, only polynomially larger propositional learning problem, i.e., learning function-free ij-determinate program clauses from ground background knowledge and ground examples is no more powerful than learning in propositional logic. The remaining advantage is that this kind of representation is more compact and therefore potentially more user-friendly in the preparation of the input.

So far, there has been no answer to the question whether the restriction to ij-determinate programs can be relaxed without the loss of polynomial computability of the ILP-problem. This section gives a negative answer to this question by proving that the ILP-problem for non depth-bounded determinate Horn clauses is PSPACE-hard and that the ILP-problem for depth bounded non-determinate Horn clauses is NP-hard.

5.1 Nonlearnability of Determinate Logic Programs of Unbounded Depth

Theorem 14 *The ILP-problem for i2-determinate Horn clauses, where i is variable, is PSPACE-Hard.*

We will prove this theorem by reducing the following PSPACE-complete problem [13] to the ILP-problem as stated above.

Definition 17 (Finite State Automata Intersection) *Given an alphabet Σ and a sequence $A_1 \ldots, A_n$ of deterministic finite state automata with input alphabet Σ, does there exist a word $w \in \Sigma^*$ accepted by each of the $A_i, 1 \leq i \leq n$?*

First let us recall some basics of deterministic finite state automata (DFA). A DFA A is formally described as a 5-tuple $(Q, \Sigma, \delta, s, F)$, where Q is the set of states, Σ is the input alphabet, $s \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is the transition function mapping $Q \times \Sigma$ to Q. The transition function δ is extended to words on Σ^* as follows: $\hat{\delta}(q, \epsilon) = q$, $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$, where ϵ is the empty word, $a \in \Sigma$, $w \in \Sigma^*$. A DFA is said to accept a word $x \in \Sigma^\star$, iff $\hat{\delta}(s, x) \in F$.

We are now able to give a useful encoding of DFAs and words as i2-determinate Horn clauses.

Definition 18 (Encoding of DFA's and words) *Let Γ be a function from $\Sigma^* \cup (Q, \Sigma, \delta, s, F)$ to Horn clauses,*
defined as follows: $\Gamma_{word}(x_1 x_2 \ldots x_n) = h(Q_0) \leftarrow x_1(Q_0, Q_1), x_2(Q_1, Q_2), \ldots, x_n(Q_{n-1}, Q_n), f(Q_n)$,
$\Gamma_{DFA}(A) = \{h(s)\} \cup \{\neg x(q, p) | x \in \Sigma, p, q \in Q, \delta(q, x) = p\} \cup \{\neg f(q) | q \in F\}$.

As is easily seen, DFAs can be encoded as ground i2-determinate Horn clauses, i.e., they are suitable as

determinate Horn clauses, i.e., they are suitable as hypotheses. The usefulness of this encoding is shown by the following lemma.

Lemma 3 *A word is recognized by a DFA iff the encoding of the word θ-subsumes the encoding of the DFA.*

Proof:

$\hat{\delta}(s,x) \in F \Rightarrow \Gamma(x) \leq_\theta \Gamma(A)$
Let $x = x_1 \ldots x_m$. Then there exists a sequence of states (the computation of A on x) $q_0, \ldots, q_m$, such that $q_0 = s, q_m \in F$, and $\delta(q_{i-1}, x_i) = q_i$. Let $\theta = \{Q_i/q_i | 0 \leq i \leq m\}$. ¿From the definition of Γ it follows that $\Gamma(x)\theta \subseteq \Gamma(A)$.

$\Gamma(x) \leq_\theta \Gamma(A) \Rightarrow \hat{\delta}(s,x) \in F$
Let $\theta = \{Q_i/q_i | 0 \leq i \leq m\}$. ¿From the definition of Γ it follows that $q_0 = s, q_m \in F$, and $\delta(q_{i-1}, x_i) = q_i$, i.e., the sequence $q_0 \ldots, q_m$ is an accepting computation of x by A. □

Now we are able to prove that Γ is indeed an encoding of the DFA intersection problem in terms of the learning problem of determinate Horn clauses.

Lemma 4 *Let $A_1 \ldots, A_n$ a set of DFA's. There exists a nonempty word $x \in \Sigma^*$ accepted by each of $A_1 \ldots, A_n$ iff there exists a linked determinate Horn clause consistent with the following set of positive and negative examples: $E^+ = \{H | H = \Gamma(A_i), 1 \leq i \leq n\}$ and $E^- =$*
$\{h(s) \leftarrow$
$f(s), x_1(s,q), \ldots, x_m(s,q), x_1(q,q), \ldots, x_m(q,q),$
$h(s) \leftarrow x_1(s,s), \ldots, x_m(s,s), x_1(q,s), \ldots, x_m(q,s)\}$,
where $\Sigma = \{x_1, \ldots, x_m\}$.

Proof:

The **if** direction
Let h be a linked determinate Horn clause consistent with the examples. As h is a generalization of all positive examples, each of its literals is of the form $h(X)$, $\neg f(X)$, or $\neg x(S1, S2)$, where $x \in \Sigma$. As h is a Horn clause $h(Q_0)$ is in h. As h is consistent with the second negative example, it contains a literal $f(Q_n)$, and as h is consistent with the first negative example, the variable Q_n is only unifiable with the variable Q_0 if this unification also unifies a second argument of a $\neg x(S1, S2)$ literal with the variable Q_0. As h is a determinate linked horn-clause, the literal $f(Q_n)$ must be linked to $h(Q_0)$ by a determinate linking chain of $\neg x(S1, S2)$ literals. Due to the second negative example any introduction of a variable in the first place of a $\neg x(S1, S2)$ literal makes the clause nondeterminate. Therefore, the determinate linking chain must be clause C_w consisting of the literal $h(Q_0)$, the literal $\neg f(Q_n)$, and the determinate linking chain $x(Q_0, Q_1), \ldots, x(Q_{n-1}, Q_n)$ is a generalization of h and thus a generalization of all positive examples. Using the argument above it is not a generalization of the negative examples. C_w is by construction the Γ encoding of a non empty word. By lemma 3 this word is accepted by each of the automata $A_1 \ldots, A_n$.

The **only if** direction
Let x be the word accepted by all the $A_1, \ldots, A_n$. By lemma 3 $\Gamma(x)$ is a generalization of E^+. $\Gamma(x)$ is not a generalization of the first negative example, as x must not be empty. $\Gamma(x)$ is not a generalization of the second negative example, as it contains a literal $\neg f(Q)$. Therefore, $\Gamma(x)$ is a consistent generalization of the examples. Clearly, $\Gamma(x)$ is linked and determinate. □

Corollary 15 *Using theorem 1 and theorem 14 we can conclude that determinate linked Horn clauses with unbounded determinate depth of terms are not PAC-learnable, as long as the widely assumed $RP \neq PSPACE$ conjecture is true.*

5.2 Nonlearnability of Non-determinate Logic Programs

Theorem 16 *The ILP-problem for 12-nondeterminate Horn clauses is NP-hard.*

We will prove this theorem by reducing the well known NP-complete SAT problem [13] to the ILP-problem as stated above. The reduction is inspired by a similar proof of Haussler [18] for *existentially quantified conjunctive expressions*.

Definition 19 (SAT) *Given a set $V = \{v_1, \ldots, v_n\}$ of Boolean variables and a set of clauses $C = \{C_1, \ldots, C_m\}$ over V, the question is whether there exists a truth assignment of V that satisfies the clauses in C.*

Now, let us prove that a SAT instance is satisfiable iff a specific ILP problem is solvable. The idea is to construct positive examples, such that every generalization consists of legal truth assignments to Boolean variables of the SAT problem and to construct the negative examples so that only hypotheses whose truth assignments satisfy the SAT problem are consistent with them.

Lemma 5 *The SAT instance (V,C) is satisfiable if and only if there exists a Horn clause consistent with $E = E^+ \cup E^-$ defined as follows: Define a set of $2n$ 1-place predicates $a_1 \ldots, a_{2n}$, each of which are mapped via Ψ to one of the $2n$ literals $v_1, \ldots, v_n, \bar{v}_1, \ldots, \bar{v}_n$ of V. The positive examples are then coded as following: $E^+ = \{p(c_i) \leftarrow$*

$p(c_i, c_{2i}), a_1(c_{2i}), \ldots, a_{n+i-1}(c_{2i}), a_{n+i+1}(c_{2i}), \ldots, a_{2n}(c_{2i})$ $| \; i : 1 \leq i \leq n\}$, *Note that* E^+ *contains* n *examples and that every example contains* $4n + 3$ *literals, i.e., the size of* E^+ *is bounded by a polynomial of the size of* V. *The negative example is coded as follows.* $E^- = \{\{p(ne)\} \cup \{\neg p(ne, ne_j), \neg a_i(ne_j) \mid \Psi(a_i) \notin C_j, 1 \leq i \leq 2n, 1 \leq j \leq m\}\}$ *Note that the example in* E^- *contains a literal for every literal in each clause in* C, *a literal for every clause in* C *and the head, i.e., the size of* E^- *is bounded by a polynomial of the size of* C.

Proof:

The **if** direction

Let h be a Horn clause that is consistent with the positive and negative examples. As h is a generalization of all positive examples, it is of the form

$$p(X) \leftarrow p(X, Y_k), a_1(Y_k), \ldots, a_{2n}(Y_k),$$

where $1 \leq k \leq 2^n$, and $\forall i : 1 \leq i \leq n$, at most one of $a_i(Y_k)$ or $a_{n+i}(Y_k)$ is in h. As h does not θ-subsume the negative example, there exists at least one k such that

$$h' \subseteq h \wedge h' = p(X) \leftarrow p(X, Y_k), a_1(Y_k), \ldots, a_{2n}(Y_k)$$

and there is no substitution $\theta = X/ne, Y_k/ne_j$ such that

$$h'\theta \subseteq \{p(ne)\} \cup \{\neg p(ne, ne_j), \neg a_i(ne_j) | \Psi(a_i) \notin C_j, 1 \leq i \leq 2n\}$$

This means that for every $C_j \in C$, h' contains at least one literal $a_i(Y_k)$, such that $\Psi(a_i) \in C_j$. Consequently, $\{\Psi(a_i) \mid a_i(Y_k) \in h'\}$ is a partial truth assignment for V that satisfies all clauses in C.

The **only if** direction

Assume that (V, C) is satisfiable. Let L be the set of literals true in an assignment that satisfies (V, C), such that either v_i or $\neg v_i$ is in L, but not both. Let h be a Horn clause defined by

$$h = \{p(X)\} \cup \{\neg p(X, Y), \neg a_i(Y) | \Psi(a_i) \in L, 1 \leq i \leq 2n\}$$

h θ-subsumes all positive examples, as for every i : $1 \leq i \leq n$ it contains either $a_i(X_1)$ or $a_{n+i}(X_1)$, but not both. Note that h is 12-nondeterminate, as every example in E^+ is 12-nondeterminate. We now prove by contradiction that h does not θ-subsume the negative example. If h θ-subsumes the negative example, there is a $\theta = \{X/ne, Y/ne_j\}$, such that

$$h\theta \subseteq \{p(ne)\} \cup \{\neg p(ne, ne_j), \neg a_i(ne_j) | \Psi(a_i) \notin C_j, 1 \leq i \leq 2n\}$$

But this means that h contains only a_i, such that $\Psi(a_i)$ is not in C_j, which contradicts the assumption that L satisfies all clauses of C. □

Let us illustrate the lemma by an example. Let $V = \{v_1, v_2\}$. We need $2n = 4$ predicates to represent the 4 possible literals that can be built from V. For readability, let us use the names v_1, v_2, $\bar{v}_1$, and $\bar{v}_2$ as predicate names, instead of a_1, a_2, a_3, and a_4. The 2 positive examples, one for each boolean variable, are coded as follows.

$$h(x) \leftarrow p(x, x_1), v_1(x_1), v_2(x_1), \bar{v}_2(x_1), p(x, x_2), \bar{v}_1(x_2), v_2(x_2), \bar{v}_2(x_2).$$
$$h(x) \leftarrow p(x, x_1), v_2(x_1), v_1(x_1), \bar{v}_1(x_1), p(x, x_2), \bar{v}_2(x_2), v_1(x_2), \bar{v}_1(x_2).$$

The least general generalization (lgg), as defined by Plotkin [35] of these positive examples is as follows:

$$h(X) \leftarrow p(X, X_1), v_1(X_1), v_2(X_1), p(X, X_2), v_1(X_2), \bar{v}_2(X_2), p(X, X_3), \bar{v}_1(X_3), v_2(X_3), p(X, X_4), \bar{v}_1(X_4), \bar{v}_2(X_4).$$

Any generalization of the positive examples must subsume the above *lgg*, i.e., be a subset of it, such as, for example: $h(X) \leftarrow p(X, X_1), v_1(X_1), v_2(X_1)$. This means the positive examples are coded in such a way, that every common generalization represents a set of legal assignments of truth values to the boolean variables. The lgg represents all the 2^n possible truth assignments for n boolean variables, i.e., it grows exponentially in the number of positive examples.

The unsatisfiable set of two clauses $C = \{\{v_1\}, \{\neg v_1\}\}$ is represented by the negative example as follows:

$$h(X) \leftarrow p(x, x_1), \bar{v}_1(x_1), \bar{v}_2(x_1), v_2(x_1), p(x, x_2), v_1(x_2), \bar{v}_2(x_1), v_2(x_2).$$

The lgg subsumes it as X_1 and X_2 of the lgg can be mapped to x_2 of the example and X_3 and X_4 of the lgg could be mapped to x_1 of the example. Therefore, no consistent hypothesis exists.

The satisfiable set of two clauses $C = \{\{v_1\}, \{v_2\}\}$ is represented by the negative example as follows:

$$h(X) \leftarrow p(x, x_1), \bar{v}_1(x_1), \bar{v}_2(x_1), v_2(x_1), p(x, x_2), \bar{v}_2(x_2), \bar{v}_1(x_1), v_1(x_2).$$

The lgg does not subsume it, as X_1 can neither be mapped to x_1 nor to x_2 of the example. Therefore, any hypothesis subsumed by $h(X) \leftarrow p(X, X_1), v_1(X_1), v_2(X_1)$ and subsuming the lgg is consistent with the examples, and represents a satisfiable truth assignment for the clauses of C.

As the negative example represents all (partial) truth assignments which make the clauses of C false, the lgg subsumes the negative example iff there is no truth assignment which makes the clauses true, i.e., if they are not satisfiable. ◇

conclude that 12-nondeterminate linked Horn clauses are not PAC-learnable, as long as the widely assumed RP ≠ NP conjecture is true.

6 Discussion

We defined an extension of the PAC-framework to include background knowledge and different inference mechanisms used in ILP. We proved some positive PAC-learnability results for determinate logic programs. We also proved some negative learnability results for classes of logic programs that are only slight extensions of the class of determinate logic programs. Taking these together, we have tried to produce a characterization of the class of logic programs that can be efficiently learned. Our positive results are of interest because several existing ILP systems use the property of ij-determinacy: GOLEM [31], FORCE2 [5], LINUS [24] and FOIL [40].

The learnability of logic programs has been studied earlier in the identification in the limit framework. Results are available in this area both for full clausal logic [36] and for definite clause logic [43, 2]. Recently, De Raedt has extended the results by Shapiro [43] towards example presentations containing clausal evidence [8, 9]. While these results are interesting, PAC-learnability results are of greater practical importance. Interest in the study of PAC-learnability of logic programs has consequently risen sharply.

While showing that sorted atoms are not PAC-learnable, Page and Frisch [33] show that a single constrained clause is PAC-learnable. Džeroski et al. [11] extend the class of PAC-learnable logic programs to k-clause ij-determinate function-free programs. However, their results hold for "simple" distributions. They also investigate the learnability of recursive determinate clauses, but their positive results for recursive programs rely on the use of queries.

An important corpus of recent research on PAC-learnability of logic programs is due to Cohen [6, 7, 5]. Using mostly the model of polynomial predictability, he characterizes several extensions of the language of ij-determinate clauses. He considers finer-grained extensions of this language than the ones considered in this paper: clauses of logarithmic, rather than unbounded depth and clauses with nondeterminacy bounded by a constant, rather than unbounded indeterminacy. A single log-depth determinate clause is not PAC-learnable. While a single clause with k free, possibly nondeterminate, variables is as hard to learn as DNF, limiting the nondeterminacy by a constant and imposing some additional restrictions yields a language that is prediction equivalent to an open prediction problem. These results pose slightly tighter bounds on the class of logic programs that can be efficiently PAC-learned from examples only, than the results in section 5.

hen's results say nothing about the polynomial solvability of the corresponding ILP-problem. Namely, there are two possible reasons for PAC non-learnability: the ILP-problem may not be polynomially solvable or more than polynomially many examples may be required for a good approximation under some probability distributions. The latter case is of limited practical relevance, as good approximations can often be achieved from much less examples than required by PAC-learning results. Our results in section 5, on the other hand, are based on proofs that the ILP-problems considered are not polynomially solvable.

While earlier results in identification in the limit, as well as the few PAC-learnability results, assume the ability to make existential and/or membership queries when learning recursive clauses, Cohen [6, 7, 5] studies the learnability of several classes of recursive logic programs from examples only. He shows that a single recursive ij-determinate clause is not PAC-learnable from examples only. The class of k-clause ij-determinate recursive logic programs, studied by Džeroski et al. [11] is also shown not be learnable from examples only. However, a class of two-clause closed linear recursive ij-determinate programs is shown to be PAC-learnable [7].

Frazier and Page [12] also investigate the learnability of several classes of recursive logic programs from examples only. They show that two-clause two-literal programs built from unary predicates, unary functions, variables and constants are polynomially PAC-learnable. Removing the restriction on predicate arity leads to non-learnability. However, if a constant bound j is placed on the arity of predicates, the corresponding class of programs becomes polynomially predictable. However, there are strong limitations on the classes of learnable recursive programs as shown in section 3.1. It is also clear that all negative results for nonrecursive clauses also hold for their recursive counterparts.

Acknowledgements

This work was partially supported by the ESPRIT BRA Project 6020 Inductive Logic Programming. This paper was written during the visit of Sašo Džeroski to the Katholieke Universiteit Leuven, Belgium, supported by the Commission of the European Communities under the grant CIPA3510920370. We would like to thank Katharina Morik and Stefan Wrobel for their comments on several drafts of the paper. We would also like to thank Jozsef Dombi, Tamas Horvath, and Gyorgy Turan, who discovered a now corrected error in lemma 4.

References

[1] M. Anthony and N.Biggs. *Computational Learning Theory.* Cambridge University Press, 1992.

guage. In *Proc. Tenth International Joint Conference on Artificial Intelligence*, pages 280–282. Morgan Kaufmann, San Mateo, CA, 1987.

[3] W. Buntine. Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36:149 – 176, 1988.

[4] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[5] W. W. Cohen. Cryptographic limitations on learning one-clause logic programs. In *Proc. Tenth National Conference on Artificial Intelligence*, pages 80–85. MIT Press, Cambridge, MA, 1993.

[6] 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, 1993.

[7] 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, 1993.

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

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

[10] S. Džeroski and N. Lavrač. Refinement graphs for FOIL and LINUS. In S.H. Muggleton, editor, *Inductive Logic Programming*, pages 319–333. Academic Press, London, 1992.

[11] S. Džeroski, S. H. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proc. Fifth ACM Workshop on Computational Learning Theory*, pages 128–135. ACM Press, New York, 1992.

[12] M. Frazier and C.D. Page. Learnability in inductive logic programming: Some results and techniques. In *Proc. Tenth National Conference on Artificial Intelligence*, pages 93–98. MIT Press, Cambridge, MA, 1993.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA, 1979.

[14] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

mation Processing Letters, 24:109–111, 1987.

[16] P. Hanschke and J. Würtz. Satisfiability of the smallest binary program. *Information Processing Letters*, 45:237–241, 1993.

[17] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's model. *Artificial Intelligence*, 36(2):177–221, 1988.

[18] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4:7–40, 1989.

[19] P. Idestam-Almquist. Generalization under implication by using or-introduction. In *Proc. European Conference on Machine Learning, ECML'93*, pages 56–64. Springer, Berlin, 1993.

[20] P. Idestam-Almquist. Recursive anti-unification. In S. Muggleton, editor, *Proc. Third International Workshop on Inductive Logic Programming, ILP'93*, pages 241–254. Jožef Stefan Institute, Ljubljana, Slovenia, 1993.

[21] J.-U. Kietz. A comparative study of structural most specific generalizations used in machine learning. In S. Muggleton, editor, *Proc. Third International Workshop on Inductive Logic Programming, ILP'93*, pages 149–164. Jožef Stefan Institute, Ljubljana, Slovenia, 1993.

[22] 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. Academic Press, London, 1992.

[23] N. Lavrač, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In *Proc. Fifth European Working Session on Learning*, pages 265–281. Springer, Berlin, 1991.

[24] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, Chichester, 1993.

[25] M. Li and P. Vitányi. Learning simple concepts under simple distributions. *SIAM Journal of Computing*, 20(5):911–935, 1991.

[26] J. W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, 2nd edition, 1987.

[27] J. Marcinkowski and L. Pacholski. Undecidability of the Hornclause implication problem. In *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 354–362. 1992.

[28] K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications.* Academic Press, London, 1993.

ming. Academic Press, London, 1992.

[30] S. H. Muggleton. Inverting implication. *Artificial Intelligence Journal*, 1993. To appear.

[31] S. H. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. First Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsha, Tokyo, 1990.

[32] T. Niblett. A study of generalization in logic programs. In *Proc. Third European Working Session on Learning*. Pitmann, London, 1988.

[33] C. D. Page and A. M. Frisch. Generalization and learnability: a study of constrained atoms. In S.H. Muggleton, editor, *Inductive Logic Programming*, pages 29–61. Academic Press, London, 1992.

[34] L. Pitt and M. K. Warmuth. Prediction-preserving reducibility. *Journal of Computer and System Sciences*, 41:430–467, 1990.

[35] G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, pages 153 – 163. American Elsevier, 1970.

[36] G. D. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, 1971.

[37] G. D. Plotkin. A further note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, pages 101 – 124. American Elsevier, 1971.

[38] K. R. Popper. *The Logic of Scientific Discovery*. Basic Books, New York, 1959.

[39] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[40] J. R. Quinlan. Knowledge acquisition from structured data – using determinate literals to assist search. *IEEE Expert*, 6(6):32–37, 1991.

[41] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23 – 41, 1965.

[42] M. Schmidt-Schauss. Implication of clauses is undecidable. *Theoretical Computer Science*, 59:287–296, 1988.

[43] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.

[44] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[45] R. Wiehagen and T. Zeugmann. Too much information can be too much for learning efficiently. In K. P. Jantke, editor, *Analogical and Inductive Inference, AII'92*, pages 72–86. Springer, Berlin, 1992.

knowledge representation. In K. Morik, editor, *GWAI-87 11th German Workshop on Artificial Intelligence*, Informatik-Fachberichte Nr. 152, pages 129–138. Springer, Berlin, October 1987.