The Appropriateness of Predicate Invention as Bias Shift Operation in ILP

IRENE STAHL

stahl@informatik.uni-stuttgart.de

Fakultät Informatik, Universität Stuttgart, Breitwiesenstr. 20-22, 70565 Stuttgart, Germany

Editor: M. des Jardins and D. Gordon

Abstract. The task of predicate invention in Inductive Logic Programming is to extend the hypothesis language with new predicates if the vocabulary given initially is insufficient for the learning task. However, whether predicate invention really helps to make learning succeed in the extended language depends on the language bias currently employed.

In this paper, we investigate for which commonly employed language biases predicate invention is an appropriate shift operation. We prove that for some restricted languages predicate invention does not help when the learning task fails and we characterize the languages for which predicate invention is useful. We investigate the decidability of the bias shift problem for these languages and discuss the capabilities of predicate invention as a bias shift operation.

Keywords: Inductive Logic Programming, Bias Shift, Predicate Invention.

1. Introduction

Because of the limited knowledge representation formalism of propositional learning algorithms and their difficulties in using substantial background knowledge, there is an increasing interest in investigating learning methods in a first order framework. *Inductive Logic Programming (ILP)* (Muggleton, 1990; Muggleton, 1993) is an approach that has received a lot of attention recently. The task of ILP is to learn logic programs by induction from examples in the presence of background knowledge.

The first order framework of ILP usually leads to an infinite hypothesis space. To allow for tractable learning procedures, it must be greatly restricted. The term *bias* refers to any basis for excluding hypotheses from the search space, other than strict completeness and consistency with the examples (Mitchell, 1980). An important part of a system's bias is the *hypothesis language*. It restricts the range of expressible concepts through the *vocabulary* to be used in the hypotheses, i.e., the available predicate, function and constant symbols, and the *syntactic form* of potential target programs.

If the intended target concept is not included in the range of expressible concepts, the hypothesis language is too restricted for the learning task. In that case, the language bias needs to be shifted. Deciding whether a learning problem fails in the given language is called the *bias shift problem* in the following, as the need for a bias shift emerges from the failure of the learning task. Shifting the bias can be done either by allowing a *less restricted form* of the hypotheses, or by extending the given vocabulary with *newly invented predicates*. This is called *predicate invention*. Both operations attempt to en-

large the hypothesis language such that it contains a complete and consistent hypothesis. However, their success depends on the current language bias.

The appropriateness of predicate invention as a bias shift operation depends on both its usefulness (i.e., the potential of making learning succeed) and the decidability of the bias shift problem for the current language bias. Only if predicate invention is useful and the bias shift problem is decidable will predicate invention be worth considering as a means to recover from a failure of the learning task. In this paper, we present utility and decidability results for a large range of language biases commonly used in ILP. These results mark the boundaries of appropriateness and feasibility for predicate invention and bias shift operations in general.

The paper is organized as follows. First, we give a formal definition of the usefulness of predicate invention in the context of ILP, and recall a general result that motivates the introduction of new predicates to overcome the limitations of the given language. In the following sections, we show for which language biases predicate invention is an appropriate shift operation and investigate the decidability of the bias shift problem for these languages. Finally, we characterize the capabilities of predicate invention as a bias shift operation, and conclude.

2. Definitions

The task of ILP is defined formally as follows. Given ground facts $E^{\oplus}$ and $E^{\ominus}$ as positive and negative examples, a logic program B as background knowledge, and a target language L , the system is to find a logic program $H \in L$ such that the following conditions hold (Muggleton & DeRaedt, 1994):

$$\begin{aligned} \text{Prior Satisfiability:} \quad & B \not\models E^{\ominus} \\ \text{Posterior Satisfiability:} \quad & B \cup H \not\models E^{\ominus 1} \\ \text{Prior Necessity:} \quad & B \not\models E^{\oplus} \\ \text{Posterior Sufficiency:} \quad & B \cup H \models E^{\oplus} \end{aligned}$$

In the following, we refer to the posterior conditions on $B \cup H$ as *consistency* and *completeness* with respect to the examples. The quadruple $(E^{\oplus}, E^{\ominus}, B, L)$ is called the *learning problem*. Whether there exists an $H \in L$ complete and consistent with respect to $E^{\oplus}$ and $E^{\ominus}$ and B is called the *bias shift problem*.

The learning problem is based on the notion of an *intended interpretation*, which captures knowledge about the truth and falsity of all ground facts in the considered domain. In the framework of identification in the limit (Gold, 1967), an infinite sequence of ground facts that are true and false in the intended interpretation are available as positive and negative examples. In this setting, the learning task is to construct a *finite axiomatization* of the intended interpretation. In more realistic scenarios only *finite* subsets of the facts true and false in the intended interpretation are given to the system. In that case, there is always a solution to the learning problem if not explicitly excluded by L , $H = E^{\oplus}$. As *trivial definitions* of that kind prevent the investigation of the utility

of any bias shift operation, we assume a mechanism similar to cross validation to exclude them from the hypothesis space.

$E^{\oplus}$ and $E^{\ominus}$ are split in training examples $E_t^{\oplus}$ and $E_t^{\ominus}$, and test examples $E_t^{\oplus}$ and $E_t^{\ominus}$. The learning algorithm is run on $E_t^{\oplus}$ and $E_t^{\ominus}$, resulting in a set of hypotheses H_i . Then, only those H_i which are complete and consistent with regard to $E_t^{\oplus}$ and $E_t^{\ominus}$ are returned. Furthermore, we assume that the hypotheses are reduced with respect to the training examples, meaning that no H_i contains redundant clauses or literals². This restriction is important because there might be learning algorithms which guess a trivial solution though they are given only a few of the examples. For example, let $E^{\oplus}$ and $E^{\ominus}$ exemplify the grandparent relation for a set of people, and B contain all parent relations among them. If L is the set of all logic programs using constants from B and the parent- and grandparent-predicates, nothing prevents a learning algorithm from hypothesizing $E^{\oplus}$ even if only a part of it is given in $E_t^{\oplus}$. However, the hypothesis $E^{\oplus}$ is not reduced with respect to $E_t^{\oplus}$ and $E_t^{\ominus}$. That is, the restriction to reduced programs excludes trivial solutions consisting, for example, of the ground facts in $E^{\oplus}$. Though this method leads to programs whose predictiveness exceeds the given examples, it is not yet satisfactory for excluding trivial definitions. This issue needs further investigation.

If the learning task fails in the target language L , L needs to be extended in order to make learning succeed. L is defined through the vocabulary, i.e., the available predicate symbols P and function symbols F , and the syntactic form S of the allowed programs. S is expressed as additional conditions on the set of well-formed formulas over P and F , $wff(P, F)$. We assume the elements of $wff(P, F)$ to be in clausal form, and P and F to be finite. The triple (S, P, F) is called the *language bias*, and we write L as $l(S, P, F)$. To extend L , each of the sets S , P or F might be extended. Extending F with new function symbols is largely unexplored. Replacing S by a superset of S , that is, allowing a less restricted form of the hypotheses, is the classical language bias shift operation. It extends L to cover a larger subset of $wff(P, F)$. In contrast, *Predicate Invention (PI)* shifts the language bias through extending P with finitely many new predicates. PI adds further hypotheses to L without violating the syntactic restrictions S .

PI, also known as *Constructive Induction*, is often described independently of the success of the learning task, as a method for extending the language of concept descriptions to allow more compact and concise expression of theories. In order to distinguish between mere compactions and predicates really missing in the vocabulary, Lapointe et al. (1993) introduced the terms *useful* and *necessary* for new predicates. Whereas useful new predicates only compress the hypothesis, necessary new predicates are needed to produce a hypothesis at all.

This differs from our framework. As we investigate PI as a bias shift operation, we deal only with necessary new predicates, i.e., predicates without which the learning task would fail. PI is considered to be *useful* if extending the target language L with finitely many new predicates makes a learning task succeed that otherwise would fail.

Definition. Let $\mathcal{L}$ be a class of first order languages. PI is *useful* in $\mathcal{L}$ if there exists a learning problem $(E^{\oplus}, E^{\ominus}, B, L)$, $L \in \mathcal{L}$, such that learning fails in $L = l(S, P, F)$, but succeeds in a language $L' = l(S, P', F) \in \mathcal{L}$, $P' \supset P$.

Proving that PI is useless for a class of languages allows the exclusion of PI as bias shift operation, prior to invoking any specific learning procedure. That is, our results capture the *a priori utility* of PI with respect to different language classes.

Our definition of usefulness is relatively weak inasmuch as only the *existence* of a learning problem that is solved using PI is required for PI to be adjudged useful. A stronger definition would demand that *every* learning problem that fails could be solved by means of PI. A theorem proved by Kleene (1952) can be interpreted as proving this strong utility of PI in the framework of identification in the limit with a first order target language.

THEOREM 1 (Kleene, 1952). *Any recursively enumerable set C of formulas in a first order language L is finitely axiomatizable in a first order language L' that extends L with finitely many additional predicate symbols.*

If C is equal to the set $E^\oplus$ of facts that are true in the intended interpretation, this theorem proves that *every* first order learning problem can be solved by inventing appropriate new predicates, provided that $E^\oplus$ is recursively enumerable. So why be concerned with the utility of PI?

There are two problems with Kleene's theorem and the strong definition of usefulness. First, the theorem states that every learning problem can be solved by PI, but does not identify which require PI to obtain a finite axiomatization. If there were none, PI could not be deemed useful despite the theorem. Second and more importantly, in the more restricted framework of ILP, both the target language $L = l(S, P, F)$ and the extended language $L' = l(S, P', F)$ are subject to the same syntactic restrictions S . PI only extends the set of available predicate symbols P , but leaves S unchanged. As there are learning tasks that fail not because of missing predicates in P , but because of the restrictions S that also apply to L' , there is no chance to prove strong results on the usefulness of PI. For example, PI is very useful for regular unary logic programs (Yardeni & Shapiro, 1991). It allows detection of recursive substructures in the examples. However, if learning fails because non-regular predicates are given as examples, PI does not help. Therefore, we adopt the weak definition of usefulness.

3. Usefulness of PI as a Bias Shift Operation

To prove that PI is useful, according to our definition, it suffices to give examples of learning problems that succeed through PI and otherwise fail. There are two different classes of hypothesis languages for which PI is useful. The first is that of infinite languages with recursive functors. It includes powerful languages, e.g., first order Horn logic. The second class contains languages restricted to a finite size by size bounds, schemes or language parameters. Though their expressiveness is restricted, they allow for tractable or even efficient learning procedures. PI mainly serves to extend the language without violating the specified parameters or schemes. It increases the expressiveness of the language without completely sacrificing its efficiency.

In the following, we briefly recall the definition of each language and present an example that proves the usefulness of PI.

3.1. Infinite Languages with Recursive Functors

Infinite languages allow us to produce infinitely many hypotheses for each learning problem. They contain at least one functor with arity ≥ 1 . PI is useful for overcoming the limitations inherent in the vocabulary of the given language, as we will show for first order Horn logic and more restricted infinite languages.

3.1.1. First Order Horn Logic

In Horn logic, all clauses in a program are restricted to contain at most one positive literal. More formally, Horn logic as the target language of a learning problem is written as $l(S, P, F)$, where

$$S = \{H \in wff(P, F) \mid \forall C \in H : C = (A \leftarrow B_1, \dots, B_m) \vee C = (\leftarrow B_1, \dots, B_m)\}$$

and F contains at least one n -ary functor, $n \geq 1$. Clauses with exactly one positive literal A are called *definite*, in contrast to *goal* clauses $(\leftarrow B_1, \dots, B_m)$. Goal clauses $(\leftarrow B)$ are often written as $\overline{B}$. Definite clauses without body literals $(A \leftarrow)$ are called *unit* clauses or *facts*. The *empty* clause is denoted by $\square$. Though first order Horn logic restricts full clausal logic, it is still very expressive. Kleene's theorem can be adapted to Horn languages so that in principle every learning problem in first order Horn logic can be solved by PI. However, as we have argued in the previous section, we still have to prove that there are learning problems that fail because of predicates missing in P and succeed through PI.

Example: Assume $E^\oplus$ and $E^\ominus$ exemplify a predicate $square(X)$ which is true for all square numbers X . Numbers are written in standard successor notation. For example, 3 is written as the third successor of zero, $s(s(s(0)))$, abbreviated $s^3(0)$. Then there is no solution to the learning problem using only the predicate $square/1$, the function $s/1$ and the constant 0.

Proof: Assume a solution H exists. Without loss of generality, we might assume that each clause in H is either of the form $square(s^{n^2}(0))$, or $square(s^m(X)) \leftarrow Body$, and $Body$ contains no existential variables nor constants. This is because body literals of the form $square(s^k(Z))$, $Z \neq X$, or $square(s^k(0))$ are always true or always false, regardless of the current proof.

That is, each non-unit clause C in H is of the form

$$square(s^m(X)) \leftarrow square(s^{k_1}(X)), \dots, square(s^{k_l}(X)), \quad k_j \neq m.$$

Assume C is called with $V = s^v(0)$, $v \geq m$. Then $k_j > m$ means that $square$ is called with V plus some constant c_j . Likewise, $k_j < m$ means that $square$ is called with V minus a constant c_j . So C can be written more informally as

$$square(V) \leftarrow square(V \pm c_1), \dots, square(V \pm c_l), \quad c_j > 0.$$

Let c be the maximum c_j for all non-unit clauses in H , and n_{max} the maximum n for all unit-clauses. Choose $w > \max\{n_{max}, c\} + 1$, and assume $H \vdash \text{square}(w^2)$ via an SLD-resolution-proof³

where $C_1, \dots, C_k \in H$. Because of the linearity of SLD-proofs, they can be written down linearly, in our case as

$$(\dots(\overline{\text{square}(w^2)} \cdot C_1\theta_1) \cdot \dots) \cdot C_k\theta_k.$$

Here, $(A \cdot C\theta)$ denotes the result of resolving the goal clause A and the definite clause C with substitution θ . As $w > n_{max}$, $C_1\theta_1$ must be of the form

$$\text{square}(w^2) \leftarrow \text{square}(w^2 \pm c_1), \dots, \text{square}(w^2 \pm c_l), \quad c_j > 0.$$

For the proof to be successful, each of the literals $\text{square}(w^2 \pm c_j)$ must be implied by H . However, as for each c_j

$$(w-1)^2 < w^2 - c_j < w^2 < w^2 + c_j < (w+1)^2$$

this means that H implies $\text{square}(u)$ for non-square numbers u , which contradicts the assumption that H is a solution to the learning problem. However, using the additional predicates *mult/3* and *add/3* with their standard definitions for integer multiplication and addition, the clause

$$\text{square}(X) \leftarrow \text{mult}(Z, Z, X)$$

solves the learning problem. □

This demonstrates that PI is capable of introducing predicates missing in the original language. In Horn logic, it is interesting to note that the new predicates must be defined recursively. If they were not, there would have been a solution to the learning problem in the original language.

A very common, though weak restriction on Horn logic is that of *connected* and *generative* clauses. The variables of connected clauses (Rouveirol, 1992; DeRaedt, 1992) must be connected to the head of the clause. A variable X in a clause $A \leftarrow B_1, \dots, B_n$ is connected if either $X \in \text{vars}(A)$ or $X \in \text{vars}(B_i)$ and B_i contains a connected variable

$Y \neq X$. Conversely, the head variables of generative clauses (Muggleton & Feng, 1990) must occur in the body, more formally $\text{vars}(A) \subseteq \text{vars}(\{B_1, \dots, B_n\})$ for a clause $A \leftarrow B_1, \dots, B_n$. As each clause in the proof of the previous example is connected and generative, it also proves the usefulness of PI for these more restricted languages.

3.1.2. Constrained Clauses

The presence of recursive functors makes PI useful even in cases where the syntactic form of the clauses is severely restricted. For example, *constrained* clauses must not contain any existential variables; more formally $\text{vars}(\{B_1, \dots, B_n\}) \subseteq \text{vars}(A)$ for each constrained clause $A \leftarrow B_1, \dots, B_n$. Because the previous example requires existential variables in its solution, another is needed to show the usefulness of PI for constrained languages. For example, there is no constrained definition of the standard *reverse*-predicate that uses only *reverse* and the list operations. But the efficient definition using *reverse_accumulate* as an auxiliary new predicate is in fact constrained.

However, if the learning problem requires existential variables to be solved, e.g., when defining *grandparent* with *parent*, PI does not help because the syntactic restrictions S remain. Constrained languages are the first example supporting the weak definition of usefulness we discussed in the previous section.

3.1.3. RUL-programs

Regular unary logic (RUL) programs (Yardeni & Shapiro, 1991) are a special case of constrained programs. They contain only unary predicates and allow non-variable argument terms only in the clause heads. The head arguments of clauses of the same predicate must have different function symbols. Additionally, every variable in a clause must occur exactly once in the head and once in the body.

The extensions of predicates defined by RUL-programs are regular sets particularly suited to describe argument types. RUL-programs allow for very efficient induction methods (Stahl et al., 1993). If the example set is regular, PI can make the learning task succeed.

Example: Let $B = \phi$,

$$E^{\oplus} = \{ t(f(g([a])), t(f(g([a, a])))) \} \quad E^{\ominus} = \{ t(g([a])), t([a]), t([]), t(g([a, a])), t([a, a]) \}$$

Then there is no complete and consistent RUL-program H using only $t/1$. Using an additional predicate symbol *newp*/1 allows a definition

$$\begin{aligned} t(f(g([a|Y])))) &\leftarrow \text{newp}(Y) \\ \text{newp}([]) & \\ \text{newp}([a|Y]) &\leftarrow \text{newp}(Y). \end{aligned}$$

□

However, if $E^\oplus$ exemplifies non-regular predicates, e.g., $E^\oplus \subseteq \{t(f(L_1, L_2)) \mid L_1, L_2 \text{ lists of the same length}\}$, only the introduction of n -ary new predicates will help.

3.2. Finite Languages

Finite languages contain for each learning problem only finitely many hypotheses. The finite size resulting from size bounds, schemes or parameters leads to restricted expressiveness when compared to infinite languages, but allows for tractable learning procedures. PI is useful for extending the language without violating the size restrictions. It increases the expressiveness of the language without sacrificing tractability.

3.2.1. Size and Complexity Measures

Heuristic size- or complexity measures place a fixed or application-dependent size- or complexity bound on the hypotheses. More formally, a hypothesis language of that kind is written as $l(S, P, F)$ with

$$S = \{H \in \text{wff}(P, F) \mid \text{size}(H) \leq \text{bound} \wedge \dots \text{additional restrictions} \dots\}$$

where $\text{size}(H)$ is the size- or complexity measure and bound is the size- or complexity bound. The additional restrictions might, for example, constrain H to a Horn formula. The intent of using size- or complexity bounds is to implement Ockham's razor principle that advocates the simplest solution to the learning problem. There are different approaches to measure the simplicity of a program. Some use only syntactic properties of the hypotheses as criteria, either independently of the examples (Muggleton & Buntine, 1988; Wrobel, 1994) or in comparison to them (Quinlan, 1990). More sophisticated measures consider the complexity of proofs derived from the theory (Muggleton, 1988; Wirth, 1989; Srinivasan et al., 1992).

Example: In CIGOL (Muggleton & Buntine, 1988) the significance of a hypothesis is measured by the degree to which it compresses the data. The size measure to be minimized is defined as

$$\begin{aligned} \text{size}(H) &= 1 + \sum_{C \in H} \text{size}(C), & H \text{ logic program} \\ \text{size}(C) &= 1 + \sum_{L \in C} \text{size}(L), & C \text{ clause} \\ \text{size}(f(t_1, \dots, t_n)) &= 2 + \sum_{i=1}^n \text{size}(t_i) \\ \text{size}(V) &= 1, & V \text{ variable} \end{aligned}$$

Then, introducing a new predicate into a program

$$\begin{aligned} H : \text{arch}(X, \text{beam}, Z) &\leftarrow \text{column}(X), \text{column}(Z) \\ &\text{arch}(X, \text{wedge}, Z) \leftarrow \text{column}(X), \text{column}(Z) \end{aligned}$$

with $\text{size}(H) = 27$ yields

$$H' : \begin{array}{l} arch(X, Y, Z) \leftarrow column(X), newp(Y), column(Z) \\ newp(beam) \\ newp(wedge) \end{array}$$

with $size(H') = 26 < size(H)$. That is, PI reduces the size of the theory and improves compression of the data. It might allow one to find a solution within the specified bounds. $\square$

Size bounds are often used to restrict the complexity of proofs derived from the target theory (Kietz & Wrobel, 1992; Kietz & Morik, 1994) in order to make both induction and deduction feasible. If the learning task fails, PI can be used to overcome the limitations without completely sacrificing feasibility.

Example: KLUSTER (Kietz & Morik, 1994) uses a restricted terminological logic as target language. In terminological logic, a concept is defined by its superconcepts and additional conditions. For example, in a drug domain, an active substance affecting excitement is defined as

$$active1 := active \wedge all(affects, excitement)$$

which transcribes to the first order sentence

$$\forall x \ active1(x) \leftrightarrow (active(x) \wedge (\forall y \ affects(x, y) \rightarrow excitement(y))).$$

KLUSTER does not allow for embedded formulas in place of *affects* and *excitement*. This restriction enables both efficient classification of objects and efficient induction of concepts, but might cause learning to fail. If, for example, the concepts *drug*, *active*, *excitement* and *pain* and the roles (relations) *contains* and *affects* are given, then the complex definition

$$sedative := drug \wedge all(contains, active \wedge all(affects, excitement))$$

is necessary to discriminate a sedative from an anodyne which affects pain. Introducing the auxiliary concept *active1* above allows one to define *sedative* without violating the syntactic restrictions of KLUSTER by

$$sedative := drug \wedge all(contains, active1).$$

However, newly defined concepts and roles increase the complexity of proofs derived from the theory. That is, the increased expressiveness comes at the price of efficiency. $\square$

To summarize, PI is useful if no solution to the learning problem exists within the specified bounds. New predicates can be employed to factor out common parts of clauses, or to express important sub-relations and exceptions in rules. The resulting theory might fit the given size and complexity bounds, enabling learning to succeed.

3.2.2. Schemes

Schemes represent one approach for describing the structure of the hypothesis clauses at an abstract level. They allow the expression of prior knowledge about the expected structure of hypothesized clauses in certain application domains. Schemes are closely related to size measures because they implicitly impose size bounds on the hypotheses. The hypothesis language is defined with respect to the schemes $\{S_1, \dots, S_n\}$ as $l(S, P, F)$ where

$$S = \{H \in \text{wff}(P, F) \mid \text{each clause in } H \text{ matches one of the schemes } \{S_1, \dots, S_n\}\}.$$

SIERES (Wirth & O'Rorke, 1991) and CAN (Tausend, 1992) use graphs as schemes to represent the number of literals and the argument dependencies between them. RDT (Kietz & Wrobel, 1992) and CIA (DeRaedt & Bruynooghe, 1992) employ function-free second-order clauses with predicate variables to describe the allowed structure of hypothesis clauses. The instantiations of the available schemes with respect to the background knowledge constitute the search space for learning.

If no solution instantiating the given schemes exists, PI may help to overcome the limitations imposed by them.

Example: Let the available scheme be

SIERES, CAN (graph)	RDT, CIA (second order clause)
<p>$S :$</p>	$S : P(X) \leftarrow Q(X), R(X)$

and let the target definition be $C : p(X) \leftarrow q(X), r(X), s(X)$. Then there is no equivalent instantiation of S . If a new predicate is introduced, the two clauses

$$\begin{aligned} p(X) &\leftarrow q(X), \text{newp}(X) \\ \text{newp}(X) &\leftarrow r(X), s(X) \end{aligned}$$

instantiate S and are equivalent to C . □

In this case, PI extends the hypothesis language without requiring more complex schemes. Additionally, it allows expression of recursive sub-relations.

Example: Suppose a set of clauses

$$\begin{aligned} p(X) &\leftarrow s(X, U), r(U) \\ p(X) &\leftarrow s(X, U), q(U, V), r(V) \\ p(X) &\leftarrow s(X, U), q(U, V), q(V, W), r(W) \end{aligned}$$

is needed to describe the target concept. Without PI, a scheme for each clause is necessary, whereas introducing a new recursively-defined predicate allows the use of the definition

$$\begin{aligned} p(X) &\leftarrow s(X, U), newp(U) \\ newp(U) &\leftarrow q(U, V), newp(V) \\ newp(U) &\leftarrow r(U) \end{aligned}$$

matching the simple schemes $P(X) \leftarrow Q(X)$ and $P(X) \leftarrow S(X, U), R(U)$. $\square$

3.2.3. Language Series

Language series (DeRaedt, 1992) are sets of parameterized languages. For each instantiation of the parameters, the resulting hypothesis language is finite. CLINT (DeRaedt & Bruynooghe, 1992) orders its parameterized languages according to increasing expressiveness. If the system fails to detect a complete and consistent definition within the current language, it shifts to a more expressive one.

As in the case of schemes, PI takes the place of shifting to a more complex language. For example, CLINT's language series 3 restricts the depth of existential quantification within the target clauses using $k + 1$ parameters $k, i_1, \dots, i_k$. Each clause C from $L_{i_1, \dots, i_k}$ must fulfill the following syntactic restrictions $S_{i_1, \dots, i_k}$:

- $head(C) = p(X_1, \dots, X_n)$, where $X_1, \dots, X_n$ are different variables,
- C is linked (= connected) and range restricted (= generative),⁴
- $body(C)$ is a subset of the inductively constructed set $B_{i_1, \dots, i_k}(X_1, \dots, X_n)$. The base case $B_i(W_1, \dots, W_m)$ contains all literals sharing at least one variable with $\{W_1, \dots, W_m\}$, and introducing at most i new variables. Then, $B_{i_1, \dots, i_k}(X_1, \dots, X_n)$ is $B_{i_k}(Z_1, \dots, Z_l)$, where $\{Z_1, \dots, Z_l\} = vars(B_{i_1, \dots, i_{k-1}}(X_1, \dots, X_n))$.

If no solution exists in the given language $L_{i_1, \dots, i_k}$, PI does the same job as shifting to a more complex language.

Example: A clause

$$C = p(X) \leftarrow r(X, U), q(U, V), p(U, W), r(V, W)$$

is not in $L_{1,0}$, but in $L_{1,1,0}$. With a new predicate it can be replaced by the two clauses

$$\begin{aligned} p(X) &\leftarrow r(X, U), newp(U) \\ newp(U) &\leftarrow q(U, V), p(U, W), r(V, W). \end{aligned}$$

in $L_{1,0}$ that are equivalent to C . $\square$

However, as in the case of schemes, PI is a more powerful operation than the pure language shift because recursive sub-relations are detectable.

Example: Looking at the last example of section 3.2.2, shifts from $L_{1,0}$ to $L_{1,1,0}$ and $L_{1,1,1,0}$ are necessary for defining the target concept if no additional predicate is introduced. In contrast, the new recursively-defined predicate allows a definition in $L_{1,0}$. $\square$

3.2.4. Determinate Clauses

The determinacy restriction (Muggleton & DeRaedt, 1994) is a semantic restriction on the number of instantiations of existentially quantified variables with respect to the background knowledge and the examples. That is, the syntax S of a determinate language $L = l(S, P, F)$ depends on B and $E^\oplus$.

Definition. Let B be a logic program and $E^\oplus$ a set of ground atoms. A clause $A \leftarrow B_1, \dots, B_n$ is *determinate* iff for every substitution θ that unifies A to a ground instance $e \in E^\oplus$ and for all $i \in \{1, \dots, n\}$, there is a unique substitution θ_i such that $(B_1 \wedge \dots \wedge B_i)\theta\theta_i$ is both ground and true in $\mathcal{M}(B \cup E^\oplus)$.⁵

In (Muggleton & Feng, 1990), determinacy is combined with a bound on both the depth of existential quantification and the number of variables that must be instantiated to make the substitution θ_i unique. In the previous section, we have shown that PI is useful to overcome these kinds of restrictions. In the following, we turn our concentration to the determinacy constraint.

If the determinacy constraint is violated, the non-determinate background knowledge $\mathcal{M}(B \cup E^\oplus)$ has to be transformed into functional form. As the functional parts of a non-determinate predicate must be named differently, PI is involved.

Example: Let $E^\oplus$ and $E^\ominus$ contain examples of the *ancestor*-relation for a set of people, e.g., *ancestor(fred, willi)*, and let B contain all *parent*-relations among them, e.g., *parent(fred, tim)* and *parent(tim, willi)*. Furthermore, we assume that each person in our domain has 0 or ≥ 2 children. Then determinate clauses contain only literals without existential variables, as only these are determinate with respect to the background knowledge and the examples. This leads to a trivial definition of the ancestor predicate and, therefore, to the failure of the learning task. To define *ancestor* with determinate clauses, the *parent*-relation in the background must be transformed into functional form, e.g., by inventing the determinate *mother* and *father* predicates. Then, the clauses

$$\begin{aligned} \text{ancestor}(\text{Anc}, \text{Desc}) &\leftarrow \text{father}(Z, \text{Desc}), \text{ancestor}(\text{Anc}, Z) \\ \text{ancestor}(\text{Anc}, \text{Desc}) &\leftarrow \text{mother}(Z, \text{Desc}), \text{ancestor}(\text{Anc}, Z) \\ \text{ancestor}(\text{Anc}, \text{Desc}) &\leftarrow \text{parent}(\text{Anc}, \text{Desc}) \end{aligned}$$

are determinate and solve the learning problem. $\square$

This kind of PI involves detecting dependencies between the arguments of a predicate and restructuring the knowledge base. It is employed in the context of inductive data engineering (Flach, 1993).

4. Uselessness of PI as a Bias Shift Operation

In spite of the general utility of PI as a bias shift operation for finite languages, there are language biases for which even PI fails to extend the range of expressible concepts. This is particularly true for function-free languages.

Proving the uselessness of PI, according to our definition, is more difficult than proving its usefulness. Instead of simply giving examples for successful applications of PI, we have to show that no extension of the target language with new predicates makes the learning task succeed. The results we prove in the subsequent sections only apply to complete algorithms. For algorithms leaving some part of the hypothesis space unexplored, e.g., hill climbing algorithms, PI may nevertheless be useful to make the learning task succeed.

4.1. Function-Free Constrained Clauses

Function-free constrained clauses are constrained clauses without any functors except for finitely many constants. More formally, a function-free constrained target language L is $l(S, P, F)$ where

$$\begin{aligned} F &= \{c_1, \dots, c_n\} \\ S &= \{H \in \text{wff}(P, F) \mid \forall C \in H : \text{vars}(\text{body}(C)) \subseteq \text{vars}(\text{head}(C))\} \end{aligned}$$

In contrast to the general case of constrained clauses, PI is useless in recovering from a learning task failure.

Example: Let examples about $\text{grandparent}(X, Y)$ be given for a set of people and let the background knowledge contain the parent -relations between them. Then, there is no non-trivial constrained program that covers the examples. Furthermore, PI is useless because it cannot introduce the necessary existential variable. $\square$

The following theorem proves this assertion.

THEOREM 2 *If there is no function-free constrained solution H to the learning problem $(E^\oplus, E^\ominus, B, L)$, $L = l(S, P, F)$, then there is also no function-free constrained solution H' in $L' = l(S, P', F)$ for each extension P' of P with finitely many new predicate symbols.*

Proof: We assume that a non-trivial solution H' in L' exists. In H' , recursive calls of new predicates are applied to permutations of the arguments of the parent goal. As there are only finitely many permutations, non-recursive definitions allowing the elimination of the new predicates in H' can be determined. This results in a non-trivial, complete and consistent H'' in L , in contradiction to the precondition of the theorem.

Construction of H'' from H' : Let

$$H' = \underbrace{C_{\text{new}}}_{\substack{\text{clauses with new} \\ \text{predicates as} \\ \text{positive literals}}} \cup \underbrace{C_{\text{def}}}_{\substack{\text{clauses that do not} \\ \text{contain a new} \\ \text{predicate positively}}}$$

Figure 1. Inductive construction of A

$$\begin{aligned} \text{Let } A_0 &= C_{\text{newp}} \\ A_{i+1} &= A_i \cup \{C \mid \exists C_1, C_2 \in A_i (C = (C_1 \cdot C_2\sigma)) \wedge \\ &\quad \neg \exists C' \in A_i (C'\theta \subseteq C)\} \end{aligned}$$

where $(C_1 \cdot C_2\sigma)$ is the result of resolving C_1 and C_2 with substitution σ , and $C'\theta \subseteq C$ standard θ -subsumption. Figure 1 shows the inductive construction of A_i for an example program

$$\begin{aligned} H' &= \text{reaches_final}(Z, X, Y) \leftarrow \text{edge}(Z, X), \text{newp}(X, Y) \\ &\quad \left. \begin{aligned} C_1 : \text{newp}(X, Y) &\leftarrow \text{edge}(X, Y), \text{newp}(Y, X) \\ C_2 : \text{newp}(X, Y) &\leftarrow \text{final}(X) \end{aligned} \right\} C_{\text{newp}} \end{aligned}$$

in a graph domain. The predicate $\text{edge}(X, Y)$ indicates a connection between the nodes X and Y , whereas $\text{final}(X)$ means that X is a final node. The first step of the inductive process constructs the clauses C_3 and C_4 . Already in the second step only clauses subsumed by those in A_1 are added. Therefore, the inductive construction stops at that point. More generally speaking, as there are only finitely many clauses reduced under θ -subsumption in each constrained function-free language L' , there is an integer n such that $A_{n+1} = A_n$. Let

$$A = A_n - \{C \in A_n \mid C \text{ contains a new predicate negatively}\}.$$

In our example, A contains C_2 and C_4 boxed in figure 1. We can show that A has the same success set as C_{newp} , that is $A \vdash a \Leftrightarrow C_{\text{newp}} \vdash a$ for each fact $a \in L'$ as follows: ' $\Rightarrow$ ': Let $A \vdash a$ be true. If a clause C used in the proof is not in C_{newp} , it has been added during the inductive construction of A , meaning that it results from resolving clauses in C_{newp} . Merging the resolution derivation of C from C_{newp} in the given proof leads to a resolution proof $C_{\text{newp}} \vdash a$.

' $\Leftarrow$ ': Let $C_{newp} \vdash a$ be true. If a clause C used in the proof is not in A , it contains new predicate literals in the body. Due to the inductive construction of A , there is a clause in A corresponding to each possibility of eliminating the new predicate literals. Therefore, there is a resolution proof $A \vdash a$.

The set A contains only non-recursively defined new predicates, the definitions of which can be used to unfold the new predicate literals in C_{def} . This results in the desired non-trivial program H'' in L . In our example, H'' is

$$\begin{aligned} reaches_final(Z, X, Y) &\leftarrow edge(Z, X), final(X) \\ reaches_final(Z, X, Y) &\leftarrow edge(Z, X), edge(X, Y), final(Y) \end{aligned}$$

meaning that a final node can be reached from Z if Z and X are connected and either X is a final node or X is linked to a final node Y . ■

Therefore, in function-free constrained Horn logic, PI is useless when the learning task fails. The same is true for the more restricted case of completely bound clauses C where $vars(head(C)) = vars(body(C))$.

4.2. Monadic Horn Logic

Monadic Horn logic is function-free Horn logic restricted to unary predicates. For a monadic hypothesis language $L = l(S, P, F)$, F is a finite set of constants and P a finite set of unary predicates. In contrast to RUL-programs, monadic logic programs need not to be constrained, but might contain existential variables. However, we can show that for each monadic logic program there is an equivalent one without existential variables.

THEOREM 3 *Given an arbitrary monadic logic program H in a language L , there exists a program H' in L with the same success set without existential variables.*

Proof: Body literals with an existential variable as argument are always true or always false, regardless of the current proof. For example, given a clause

$$bird(X) \leftarrow has_beak(X), flies(Z),$$

the literal $flies(Z)$ is true if there is some arbitrary flying object in the theory, and false otherwise. The instantiation of X in the current proof is irrelevant for the truth or falsity of $flies(Z)$. Therefore, literals of that form can be eliminated from H . For each clause $C_i \in H$, let X_i be the set of literals in C_i that contain existential variables. If $H \vdash X_i$, add $C_i - X_i$ to H' . Then H' has the same success set as H , that is $H' \vdash a \Leftrightarrow H \vdash a$ for each fact $a \in L$.

' $\Rightarrow$ ': For each clause $C'_i \in H'$ used in the proof of a there is a clause $C_i \in H$, $C'_i \subseteq C_i$. As $H \vdash (C_i - C'_i)$, there is also a proof $H \vdash a$.

' $\Leftarrow$ ': For each clause $C_i \in H$ used in the proof of a there is a clause $C'_i \in H'$, $C'_i \subseteq C_i$, otherwise parts of C_i would not have been eliminated. Therefore, there is also a proof $H' \vdash a$. ■

So, only constrained clauses need to be considered when learning in monadic Horn logic. Therefore, PI is useless if learning fails.

4.3. Function-Free Horn Logic

Function-free logic programs contain no functors except for finitely many constants. That is, the set F of a function-free language $L = l(S, P, F)$ is a finite set $\{c_1, \dots, c_k\}$ of constants. Excluding arbitrary functors leads to the decidability of logical entailment, in contrast to full first order Horn logic.

This decidability accounts for the uselessness of PI. In section 5.2 we prove that, given $(E^\oplus, E^\ominus, B, L)$ with a function-free language L , only clauses with at most n different variables need to be considered for the target program H . The parameter n depends on the number of constants in B and L , and the arity of the available predicates. This property accounts for the decidability of the learning problem on the one hand, and the uselessness of PI on the other. As only clauses with up to n different variables need to be considered, new recursively-defined predicates can be eliminated by a method similar to that for constrained clauses.

THEOREM 4 *If there is no function-free solution H to the learning problem $(E^\oplus, E^\ominus, B, L)$, $L = l(S, P, F)$, then there is also no function-free solution H' in $L' = l(S, P', F)$ for each extension P' of P with finitely many new predicate symbols.*

Proof: As in the proof of theorem 2, we assume that a complete and consistent H' in L' exists, and construct a complete and consistent H'' in L , in contradiction to the precondition of the theorem. The crucial difference from theorem 2 is in the inductive construction of the set A of non-recursive new predicate definitions:

$$\begin{aligned} A_0 &= C_{newp} \\ B_{i+1} &= A_i \cup \{C \mid \exists C_1, C_2 \in A_i (C = (C_1 \cdot C_2\sigma)) \wedge \\ &\quad \neg \exists C' \in A_i (C'\theta \subseteq C)\} \end{aligned}$$

B_{i+1} might contain clauses with $> n$ variables, where n is the bound on the number of variables. By the method we describe in section 5.2, an extensionally equivalent set A_{i+1} of clauses with at most n variables is constructed from B_{i+1} . Because there are only finitely many clauses reduced under θ -subsumption with $\leq n$ variables, there is an integer k such that $A_{k+1} = A_k$. The set

$$A = A_k - \{C \in A_k \mid C \text{ contains a new predicate negatively}\}$$

can be proved to be extensionally equivalent to C_{newp} . Since it contains only non-recursive definitions of new predicates, it can be used to unfold the new predicates in C_{def} , resulting in the desired program H'' . ■

So, if the learning method fails to find a function-free solution, PI is useless in recovering from the failure. However, this result must not be over-generalized. It does

not apply if incomplete search is used or if the background knowledge is changing. For example, if a definition of *male_ancestor* is to be induced that is valid no matter which family tree is given as background knowledge and examples, introducing an auxiliary predicate *ancestor* is useful. But if both the background knowledge and the examples are fixed, PI is useless in recovering from a failure of the learning task.

5. Decidability of the Bias Shift Problem

Bias shift operations are applied to recover from a failure of the learning task. Deciding when a bias shift is necessary involves deciding whether the learning task fails in the given language. For a bias shift operation like PI to be feasible for a class of languages, both its a priori utility and the decidability of the bias shift problem itself need to be ensured.

Similar to the utility results of the previous sections, the decidability of the bias shift problem depends on the complexity of the target language. Two central results, the undecidability of the bias shift problem for first order Horn logic and the decidability for function-free Horn logic, mark the boundaries of feasibility for bias shift operations.

5.1. Undecidability for First Order Horn Logic

Though first order Horn logic is less expressive than full clausal logic, it still does not allow for deciding logical entailment. Accordingly, the bias shift problem is undecidable in that framework.

THEOREM 5 *The bias shift problem is undecidable for learning first order Horn theories.*

Proof: Suppose the bias shift problem is decidable. Then the problem of deciding logical entailment, an undecidable problem, would be decidable.

Let P be a logic program and e a ground fact. To decide whether $P \vdash e$ holds, it suffices to decide whether the bias has to be shifted for the learning problem consisting of $E^{\oplus} = \{e\}$, $E^{\ominus} = \emptyset$, $B = \emptyset$ and $L = \{P\}$. ■

Though PI is useful in the framework of first order Horn logic, theorem 5 shows that it is not feasible. The language biases discussed can be seen as attempts to approximate the decision on shifting the bias and introducing new predicates in first order Horn logic.

5.2. Decidability for Function-Free Horn Logic

The restriction to finitely many constants is fundamental for function-free languages, otherwise they are as expressive as unrestricted Horn logic. This leads to an interesting observation, when inductive inference is concerned. If all n constants in B , $E^{\oplus}$ and $E^{\ominus}$ are known, it suffices to consider clauses with at most n variables for the target program.

In that case, the bias shift problem is decidable because the finitely many programs can be enumerated and tested for completeness and consistency. The following theorem captures the above observation.

THEOREM 6 *Given a function-free language $L = l(S, P, \{c_1, \dots, c_n\})$, then for each H in L there exists an H' in L such that all clauses in H' contain at most n variables, and $H \vdash a$ iff $H' \vdash a$ for each fact a in L .*

Proof: (Reinhardt, 1993) Each clause $C \in H$ with $m > n$ variables is replaced by n^m clauses $C\sigma$ for each possible substitution $\sigma : \text{vars}(C) \rightarrow \{Z_1, \dots, Z_n\}$ substituting the variables of C with at most n different new variables. For example, if C is

$$p(X, Y) \leftarrow q(X, Z), r(Z, Y)$$

and $n = 2$, we get $2^3 = 8$ different clauses $C\sigma$, e.g.

$$\begin{aligned} p(Z_1, Z_1) &\leftarrow q(Z_1, Z_1), r(Z_1, Z_1) \\ p(Z_1, Z_1) &\leftarrow q(Z_1, Z_2), r(Z_2, Z_1) \\ p(Z_1, Z_2) &\leftarrow q(Z_1, Z_1), r(Z_1, Z_2) \\ &\dots \end{aligned}$$

For the resulting program H' we have to show that $H \vdash a \Leftrightarrow H' \vdash a$.

' $\Rightarrow$ ': Without loss of generality we assume $H \vdash a$ via an SLD-proof

$$((\dots((\bar{a} \cdot C_1\theta_1) \cdot C_2\theta_2)\dots) \cdot C_k\theta_k).$$

Then, $((\dots((\bar{a} \cdot C_1\theta_1 \dots \theta_k) \cdot C_2\theta_2 \dots \theta_k)\dots) \cdot C_k\theta_k)$ is also a proof that $H \vdash a$. Given a substitution ρ_i which substitutes for all variables in $C_i\theta_i \dots \theta_k$ an arbitrary constant, $((\dots((\bar{a} \cdot C_1\theta_1 \dots \theta_k\rho_1) \cdot C_2\theta_2 \dots \theta_k\rho_2)\dots) \cdot C_k\theta_k\rho_k)$ is also a proof. Now $C_i\theta_i \dots \theta_k\rho_i$ is a ground clause with at most n different constants so that there exists a $C'_i \in H'$ and a substitution ρ'_i such that $C_i\theta_i \dots \theta_k\rho_i = C'_i\rho'_i$. Thus, $((\dots((\bar{a} \cdot C'_1\rho'_1) \cdot C'_2\rho'_2)\dots) \cdot C'_k\rho'_k)$ is a proof.

' $\Leftarrow$ ': We assume $H' \vdash a$ via an SLD-proof $((\dots((\bar{a} \cdot C'_1\theta'_1) \cdot C'_2\theta'_2)\dots) \cdot C'_k\theta'_k)$. For each C'_i there is a $C_i \in H$, either $C_i = C'_i$ if C_i contains $\leq n$ variables, or $C_i\sigma_i = C'_i$. Thus, $((\dots((\bar{a} \cdot C_1\sigma_1\theta'_1) \cdot C_2\sigma_2\theta'_2)\dots) \cdot C_k\sigma_k\theta'_k)$ is a proof that $H \vdash a$.

An alternative proof sets H' to the minimal model of H , $\mathcal{M}(H)$ ⁵. Then H' trivially contains clauses with at most n variables. However, this proof technique violates the restriction to non-trivial programs which our proof preserves. ■

The crucial question for the decidability of the bias shift problem in function-free languages is whether all constants in the learning problem are known during induction. This will generally not be the case, especially if techniques to exclude trivial definitions are used. These techniques present only some of the examples as training set to the learning method. The number of new constants in the remaining test set is unknown, and likewise the upper bound for the number of different variables in the target clauses.

However, a closer investigation of the situation when $E^\oplus$ and $E^\ominus$ contain constants outside B and L shows that the result of theorem 6 can be generalized. The generalization is based on the subsumption theorem (Rouveirol, 1991). A program H implies

a ground fact e with constants outside H if and only if H implies the fact e' that results from replacing all new constants in e by variables. That is, the unknown constants themselves do not matter for the inductive inference method, rather the potential number of occurrences within one specific example. This number is bounded by the maximum predicate arity $\max_A$ in $E^\oplus$ and $E^\ominus$. Thus, given a learning problem $(E^\oplus, E^\ominus, B, L)$ where L and B are missing some of the constants in $E^\oplus$ and $E^\ominus$, it suffices to consider clauses with at most $n + \max_A$ variables for the target program. So even in cases where the target program is to handle examples with unknown, new constants, the bias shift problem is decidable.

An interesting question is whether it is really desirable to induce programs that cover examples with arbitrary constants. Due to the subsumption theorem, covering ground facts with new constants means, in fact, that the corresponding universally quantified formula is implied. Often, this is so strong that many systems require knowledge about all constants in $E^\oplus$ and $E^\ominus$ to be present in B , e.g., (Quinlan, 1990; DeRaedt & Bruynooghe, 1989). Moreover, in cases where examples contain constants missing in B , techniques to acquire background knowledge about them have been proposed (DeRaedt et al., 1991).

However, this technique leads to a stronger success criterion for learning. The induced program is to cover not only the given examples with respect to the given background knowledge, but arbitrary examples with respect to an augmented background knowledge. Although this criterion makes PI useful, it violates the restriction to function-free programs because of the need to consider potentially infinitely many constants.

5.3. Decidability for Finite Languages

The languages for which we have proved the usefulness of PI in section 3.2 are all finite languages, i.e., for a learning problem $(E^\oplus, E^\ominus, B, L)$ there are only finitely many hypotheses in L . Additionally, if the completeness and consistency tests of hypothesized programs are decidable or restricted to decidable cases, the bias shift problem can in principle be solved by enumerating and testing the finitely many programs.

For example, given function-free $E^\oplus$, $E^\ominus$ and B , there are only finitely many programs in each language of language series 3 (DeRaedt, 1992). Moreover, because of the lack of recursive functors, the completeness and consistency tests are decidable. Therefore, the bias shift problem is decidable.

In contrast, the determinacy restriction is not sufficient to ensure the decidability of the bias shift problem. If an unrestricted background knowledge B is given, both the completeness and consistency tests might be undecidable and there might be infinitely many determinate programs. In GOLEM (Muggleton & Feng, 1990), B is replaced by its finite Herbrand h -easy model $\mathcal{M}_h(B)$, the set of ground atoms derivable from B in at most h resolution steps. This restricts L to a finite language, and renders the completeness and consistency tests decidable.

Figure 2. Usefulness of PI and decidability of the bias shift problem

6. Capabilities of PI as Bias Shift Operation

Figure 2 summarizes the decidability and utility results of the previous sections. The languages for which PI is a useful bias shift operation can be divided in two classes. The first is unrestricted or weakly restricted first order Horn logic. If learning fails in a language of that kind, predicate invention helps to overcome the limitations of the given vocabulary. PI really has the capability to introduce new predicates missing in the original vocabulary. However, this power comes at the price of the undecidability of the bias shift problem.

The second class of languages for which PI is useful contains languages restricted to a finite size by language parameters, schemes or size bounds. For each learning problem, these languages result in a finite hypothesis space. In that framework, new predicates mainly serve the task of extending the given finite language without violating the specified parameters or schemes. They do the same job as shifts to a more general language, e.g., in CLINT (DeRaedt & Bruynooghe, 1992) or supplying more complex schemes. In the strict logical sense, most of these predicates are not necessary because they can be eliminated by unfold-operations. However, in contrast to the pure language shifts, new

predicates additionally allow for expressing recursive sub-relations. Therefore, PI is a more powerful bias shift operation.

7. Conclusions

The central aim of PI in ILP is to extend the given vocabulary if it is insufficient for the learning task. This paper investigates the utility of PI as a bias shift operation and the decidability of the bias shift problem.

Our results are largely negative for PI. For function-free languages, PI can be proved as useless. These languages are restricted so strongly that PI cannot increase their expressiveness. For first order Horn logic, PI is useful and really capable of introducing predicates missing in the original language. But inductive inference is unfeasible in this framework.

Only for the remaining class of finite languages can PI be shown to be useful. However, this is just the class of languages for which inductive inference can (partly) be shown to be tractable, as first PAC-learning results show, e.g., (Cohen, 1993; Dzeroski et al., 1992). Furthermore, PI is useful both for syntactically shifting the language bias and for extending the vocabulary with necessary new predicates. So, PI integrates two different bias shift operations in one. This leads us to the supposition that PI is the most promising approach to realize bias shift operations in real-world ILP systems with their finite hypothesis languages. It can be integrated more naturally than, for example, language series and we suppose that, as a more goal-driven operation, PI involves less search than syntactic bias shift operations. Thus, despite the negative results for unrestricted languages, PI offers a powerful and feasible way out of the contradiction between two requirements of real-world ILP-systems: expressiveness and efficiency (Kietz & Morik, 1994).

Acknowledgments

This work has been supported by the European Community ESPRIT BRA 6020 ILP (Inductive Logic Programming). I want to thank Birgit Tausend, Peter Forster and Rüdiger Wirth for reading and commenting on earlier drafts, and Klaus Reinhardt for his inspiring proof ideas. The comments of the anonymous reviewers and particularly their help in clarifying and shortening the proof of theorem 5 have also greatly helped to improve the article. I am especially grateful to Teri Kinealy and Rosanne Price for proof-reading the manuscript.

Notes

1. In contrast to our setting, (Muggleton & DeRaedt, 1994) assume that $E^\ominus$ contains negative evidence, in our case negated ground atoms. This requires to rewrite the satisfiability conditions to $B \cup E^\ominus \not\models \square$ and $B \cup H \cup E^\ominus \not\models \square$, respectively.

2. A clause C is redundant in H with respect to $E_l^\oplus$ and $E_l^\ominus$ if $B \cup (H - \{C\}) \vdash E_l^\oplus$ and $B \cup (H - \{C\}) \not\vdash E_l^\ominus$. A literal l is redundant in a clause $C \in H$ if the program H' that results from H by replacing C with $C - \{l\}$ is still complete and consistent with respect to the examples.
3. SLD stands for linear resolution with selection function for definite clauses.
4. These terms are more usual in the deductive data base literature.
5. $\mathcal{M}(T)$ is the least Herbrand model of T , i.e., the set of all ground atoms constructed using functors and predicate symbols in T that are logically entailed by T .

References

- Cohen, W. (1993). PAC-learning a restricted class of recursive logic programs. In *Proc. of the 3rd International Workshop on Inductive Logic Programming*.
- De Raedt, L. (1992). *Interactive theory revision: an inductive logic programming approach*. Academic Press.
- De Raedt, L. & Bruynooghe, M. (1989). Towards friendly concept learners. In *Proc. of IJCAI*.
- De Raedt, L. & Bruynooghe, M. (1992). Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150.
- De Raedt, L., Feyaerts, J. & Bruynooghe, M. (1991). Acquiring object-knowledge for learning systems. In Y. Kodratoff, editor, *Proc. of the Fifth European Working Session on Learning*. Springer.
- Dzeroski, S., Muggleton, S. & Russel, S. (1992). PAC-learnability of determinate logic programs. In *Proc. of the 5th ACM Workshop on Computational Learning Theory*.
- Flach, P.A. (1993). Predicate invention in inductive data engineering. In *Machine Learning: ECML-93, European Conference on Machine Learning, Wien, Austria*. Springer.
- Gold, E.M. (1967). Language identification in the limit. *Information and Control*, 10:447–474.
- Kietz, J.U. & Wrobel, S. (1992). Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press.
- Kietz, J.U. & Morik, K. (1994). A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14:193–217.
- Kleene, S.C. (1952). Finite axiomatizability of theories in the predicate calculus using additional predicate symbols. In *Two Papers on the Predicate Calculus*, number 10 in *Memoirs of the American Mathematical Society*.
- Lapointe, S., Ling, C. & Matwin, S. (1993). Constructive inductive logic programming. In *Proc. of the IJCAI-93*. Morgan Kaufmann.
- Mitchell, T.M. (1980). The need for biases in learning generalizations. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann.
- Muggleton, S. (1988). A strategy for constructing new predicates in first order logic. In *Proceedings of the Third European Working Session on Learning*. Pitman.
- Muggleton, S. (1990). Inductive logic programming. In *First Conference on Algorithmic Learning Theory*, Tokio, Ohmsha.
- Muggleton, S. (1993). Inductive logic programming: Derivations, successes and shortcoming. In *Machine Learning: ECML-93, European Conference on Machine Learning, Wien, Austria*. Springer.
- Muggleton, S. & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Fifth International Conference on Machine Learning*. Morgan Kaufmann.
- Muggleton, S. & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, Special Issue on 10 Years of Logic Programming.
- Muggleton, S. & Feng, C. (1990). Efficient induction of logic programs. In *First Conference on Algorithmic Learning Theory*, Tokyo, Ohmsha.
- Quinlan, J.R. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239–266.
- Reinhardt, K. (1993). Personal communication.
- Rouveirol, C. (1991). *ITOU: Induction de Théories en Ordre Un*. PhD thesis, Université Paris Sud, Centre d'Orsay.
- Rouveirol, C. (1992). ITOU: Induction of first order theories. In Muggleton, S., editor, *Inductive Logic Programming*. Academic Press.

- Srinivasan, A., Muggleton, S., & Bain, M. (1992). Distinguishing exceptions from noise in non-monotonic learning. In *Proceedings of ILP'92, Tokyo*.
- Stahl, I., Tausend, B., & Wirth, R. (1993). Two methods for improving inductive logic programming systems. In *Machine Learning: ECML-93, European Conference on Machine Learning, Wien, Austria*. Springer.
- Tausend, B. (1992). Using and adapting schemes for the induction of horn clauses. In *ECAI Workshop Logical Approaches to Machine Learning*, Wien.
- Wirth, R. (1989). *Lernverfahren zur Vervollständigung von Hornklauselmengen durch inverse Resolution*. Dissertation, Fakultät Informatik, Universität Stuttgart.
- Wirth, R. & O'Rorke, P. (1991). Constraints on predicate invention. In *Eighth International Conference on Machine Learning*. Morgan Kaufmann.
- Wrobel, S. (1994). Concept formation during interactive theory revision. *Machine Learning*, 14:169–191.
- Yardeni, E. & Shapiro, E. (1991). A type system for logic programs. *Journal of Logic Programming*, (10):125–153.

Received November 18, 1993

Final Manuscript August 22, 1994