On the Utility of Predicate Invention in Inductive Logic Programming

Irene Stahl

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

Abstract. The task of predicate invention in ILP is to extend the hypothesis language with new predicates in case that 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 bias that is 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 in case that the learning task fails, and characterize the languages for which predicate invention is useful as bias shift operation.

1 Introduction

Inductive logic programming (ILP) [Mug92] aims to learn logic programs from examples in the presence of background knowledge. As opposed to propositional frameworks, this setting leads to a generally infinite space of possible target programs systems have to consider.

Biases are used to search and restrict this hypothesis space. *Algorithmic* biases guide the search of a system, whereas *absolute* biases restrict the space of potential solutions that is considered at all. The absolute bias determines the target language of inductive inference by constraining the *vocabulary* to be used in hypotheses, that is the available predicate, function and constant symbols, and the *form* and *complexity* of potential target programs. The aim of using a language bias is to consider fewer, ideally only finitely many well-structured or understandable hypotheses.

However, the restrictions imposed by the absolute bias might be too strong such that the hypothesis space does not include a correct target program. In that case, it needs to be relaxed such that the enlarged hypothesis space contains a solution. Extending the given vocabulary with *newly invented predicates*, for short *predicate invention (PI)*, is one possibility to shift the bias. Besides overcoming the limitations of the insufficient vocabulary, the new predicates might also allow for the formulation of simpler hypotheses that conform to the complexity restrictions of the target language.

However, the appropriatness of PI depends on its prior utility for the current language bias. In some languages it is the only possibility to make learning succeed, whereas in others it is useless in case that the learning task fails.

In this paper, we investigate the utility of PI as bias shift operation in ILP. We give a formal definition of the usefulness of PI with respect to the absolute bias, 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 PI is an appropriate shift operation. Finally, we characterize the languages for which PI is useful, and the contribution of the new predicates to the resulting target programs.

2 Definitions

The task of ILP is defined formally as follows. Given ground facts $E^{\oplus}$ and $E^{\ominus}$, the positive and negative examples, a logic program B as background knowledge, and a target language L, the system is to find a logic program $P \in L$ such that $B \cup P \vdash E^{\oplus}$ (*completeness*) and $B \cup P \nvdash E^{\ominus}$ (*consistency*). The quadruple $(E^{\oplus}, E^{\ominus}, B, L)$ is called the *learning problem*.

It is based on an *intended interpretation* of the user that satisfies at least B and $E^{\oplus}$. In the limit, all ground facts true and false in the intended interpretation are given as positive and negative examples. In this setting, the learning task means to construct a *finite axiomatisation* of the intended interpretation. In more realistic scenarios $E^{\oplus}$ and $E^{\ominus}$ are *finite* subsets of the facts true and false in the intended interpretation. In that case there is always a solution to the learning problem if not explicitly excluded by L, namely $P = E^{\oplus}$. As these *trivial definitions* prevent the investigation of the utility of any bias shift operation, we assume some mechanism to exclude them from the hypothesis space, as e.g. cross validation. This technique splits $E^{\oplus}$ and $E^{\ominus}$ in training and test examples. The target program is constructed from the training examples, and verified on the test examples. Though cross validation leads to programs the predictiveness of which exceeds the given training examples, it is not completely satisfactory for excluding trivial definitions. This issue needs a further, more thorough investigation.

If a learning task fails in the given language L, L is too restrictive to finitely axiomatize the intended interpretation. PI is useful if extending L with finitely many new predicates makes a learning task succeed that fails otherwise. Utility is defined with respect to the class of target languages, that is the absolute bias[1].

Definition 1. 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, but succeeds in a language $L' \in \mathcal{L}$ that extends L with finitely many new predicates.

Definition 1 is relatively weak in as much as only the *existence* of a learning problem that succeeds through PI is required for PI to be useful. A stronger definition would demand that *every* learning problem could be solved by means of PI. Kleene [Kle52] has proved this strong utility of PI in the framework of identification in the limit with unrestricted first order logic as target language.

[1] A separate problem we do not investigate in this paper is the utility of PI with respect to the shift of algorithmic biases.

Theorem 2. *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 set 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.

However, in the more restricted framework of ILP this strong view of usefulness is unsuitable because both the target language L and the extended language L' are subject to the same restrictions. As there are learning tasks that fail not because of missing predicates in L, but because of the restrictions that also apply to each extension L', there is no chance to prove strong usefulness results. For example, in section 3.7 we show that PI is very useful for regular unary logic programs [YS91]. It allows to detect 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 1 of usefulness.

3 Usefulness Results

To prove usefulness results according to definition 1 it suffices to give examples of learning problems that succeed through PI and fail otherwise. There are two different classes of hypothesis languages for which PI is useful. The first is unrestricted or weakly restricted Horn logic. The second class contains languages restricted to a fixed size by size bounds, schemes or language parameters. Here, PI mainly serves to extend the language without violating the specified parameters or schemes.

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

3.1 Unrestricted and Weakly Restricted Horn Logic

In Horn logic, all clauses are restricted to contain at most one positive literal. Though this restricts full clausal logic, first order Horn logic is still very expressive. Accordingly, logical implication is undecidable as in case of first order logic.

PI is useful to recover from the failure of a learning task $(E^{\oplus}, E^{\ominus}, B, L)$ for Horn clause languages L. For example, given facts about the multiplication of natural numbers, there is no solution of the learning problem using only the predicate *multiply*/3, the function $s/1$ and the constant 0 (cf. [Lin91]). Only introducing a new predicate *add* will make learning succeed.

The usefulness of PI as bias shift operation is passed on to weak restrictions of Horn logic, namely *connected* and *generative* clauses. The body variables of connected clauses [Rou91, Rae91] must be related to the head of the clause. Vice versa, the head variables of generative clauses [MF90] must occur in the body. As each arbitrary clause can be made connected, respectively generative, by adding body literals, neither of both is a real restriction. Therefore, inducing connected or generative clauses is as hard as inducing arbitrary clauses. Likewise, PI is useful as bias shift operation.

3.2 Size and Complexity Measures

Heuristic size- or complexity measures place a fixed or application-dependent size- or complexity bound on the hypotheses. They realize Ockham's razor principle that prefers the simplest complete and consistent program. There are different approaches to measure the simplicity of a program. Some use only syntactic properties of the hypotheses as criterion, either independently of the examples [MB88, Wro] or when compared to them [Qui90]. Others take into account the complexity of proofs from the theory [Sha83, Mug88, Wir89, SMB92, MSB92].

PI is useful if no solution of the learning problem exists within the specified bounds. New predicates can be employed to factor out common parts of clauses, or to express recursive subrelations and exceptions intensionally. The resulting theory might fit the given size and complexity bounds and make learning succeed.

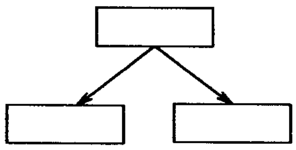
3.3 Schemes

Schemes describe the structure of the hypothesis clauses at an abstract level. They allow to express prior knowledge about the expected structure of hypothesis clauses in certain application domains, for example DCG clauses for grammar learning. As only finitely many schemes are given, the search space is finite.

SIERES [Wir91a] and CAN [Tau92] use graphs to represent the number of literals and the argument dependencies between them. RDT [Kie91] and CIA [Rae92] 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 helps to overcome the limitations imposed by them.

Example 1. Let the available scheme be

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

and let the target definition be

$$C \; : \; p(X) \leftarrow q(X), r(X), s(X)$$

Then there is no complete and consistent instantiation of S. If a new predicate is introduced, two clauses instantiating S that are equivalent to C can be found:

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

That is, PI extends the hypothesis language without requiring more complex schemes. Additionally it allows to express recursive subrelations.

3.4 Language Series

Language series [Rae92, Rae91] are sets of parametrized languages. For each instantiation of the parameters the resulting hypothesis language is finite. CLINT [Rae92] orders its parametrized languages according to generality. If the system fails to detect a complete and consistent definition within the current language, it shifts to a more general one.

As in the case of schemes, PI is useful to replace the shift to a more complex language. For example, CLINT's language series 3 restricts the depth of existential quantification within the target clauses:

$$\begin{aligned} \forall i_1, .., i_k \geq 0 \ : \ L_{i_1,..,i_k} = \{ \ C \mid & \ head(C) = p(X_1, .., X_n), \ X_j \neq X_k \ \forall \ j \neq k \\ & \wedge \ body(C) \subseteq B_{i_1,..,i_k}(X_1, .., X_n) \\ & \wedge \ C \text{ linked and range restricted}^2 \ \} \\ \text{where } B_i(X_1, .., X_n) = \{ \ q(Y_1, .., Y_k) \mid & \ |\{Y_1, .., Y_k\} - \{X_1, .., X_n\}| \leq i \ \wedge \\ & \ \{Y_1, .., Y_k\} \cap \{X_1, .., X_n\} \neq \phi \ \} \\ B_{i_1,..,i_k}(X_1, .., X_n) = \{ \ q(Y_1, .., Y_k) \in & \ B_{i_k}(Z_1, .., Z_l) \mid \\ & \{Z_1, .., Z_l\} = vars(B_{i_1,..,i_{k-1}}(X_1, .., X_n)) \ \} \end{aligned}$$

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

Example 2. A target 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 rewritten to two clauses in $L_{1,0}$:

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

However, as in the case of schemes PI is a more powerful operation than the pure language shift, as recursive subrelations might be detected.

[2] Both linked and connected, and range restricted and generative, are synonymous. The formers are more usual in the deductive data base literature.

Example 3. Suppose a set of clauses

$$\begin{array}{l} p(X) \leftarrow s(X,U), r(U) \\ p(X) \leftarrow s(X,U), q(U,V), r(V) \\ p(X) \leftarrow s(X,U), q(U,V), q(V,W), r(W) \\ \quad\quad\quad\quad \ldots\ldots \end{array}$$

is needed to describe the target concept. Without PI, shifts from $L_{1,0}$ to $L_{1,1,0}$ and $L_{1,1,1,0}$ are necessary, whereas introducing a recursively defined new predicate allows a definition in $L_{1,0}$:

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

3.5 Determinate Clauses

Similar to language series 3 of CLINT, the determinacy restriction [MF90] is used to constrain the maximum depth of existential quantification. Additionally, a semantic restriction is placed on the the number of instantiations of the existentially quantified variables with respect to the background knowledge.

Definition 3. Let B be a logic program and $E^{\oplus}$ a set of ground atoms. Every unit clause is *0j-determinate*. An ordered clause $A \leftarrow B_1, .., B_m, B_{m+1}, .., B_n$ is *ij-determinate* iff $A \leftarrow B_1, .., B_m$ is $(i-1)j$-determinate, and every literal $B_k \in \{B_{m+1}, .., B_n\}$ contains only determinate terms and has degree at most j. A term t found in B_k is *determinate* with respect to B_k iff for every substitution θ such that $A\theta \in E^{\oplus}$ and $\{B_1, .., B_m\}\theta \subseteq \mathcal{M}(B)$[3] there is a unique ground atom $B_k\theta\sigma \in \mathcal{M}(B)$. The *degree* of B_k with respect to t is the number of variables in B_k which must be instantiated to determine t.

In case that no ij-determinate solution of the learning problem exists, PI is useful to overcome the limitations of ij-determinacy, at least for the parameter i.

Example 4. Given functional predicates $q(+,-)$, $r(+,-)$ and $s(+,-)$, the clause

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

violates the constraint $i = 2$ as the variable W is found at depth $i = 3$. With a new predicate it can be rewritten to two $2j$-determinate clauses:

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

However, in case that the j-parameter is violated, the situation is more complicated. In general, PI is not capable of decreasing the degree of literals from $k > j$ to $\leq j$.

[3] $\mathcal{M}(B)$ is the set of all ground atoms derivable from B.

Example 5. Given functional predicates $q(+,-)$, $r(+,-)$ and $t(+,+,-)$, the clause

$$C = p(X,Y) \leftarrow q(X,N1), r(X,N2), t(N1,N2,Y)$$

violates the constraint $j = 1$ in the last literal. It cannot be rewritten in $i1$-determinate clauses.

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

Example 6. The clause

$$ancestor(Anc, Desc) \leftarrow \underline{parent(Z, Desc)}, ancestor(Anc, Z)$$

is not determinate because of the *parent*-literal. In order to express *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

$$\begin{array}{l} ancestor(Anc, Desc) \leftarrow father(Z, Desc), ancestor(Anc, Z) \\ ancestor(Anc, Desc) \leftarrow mother(Z, Desc), ancestor(Anc, Z) \end{array}$$

are determinate and equivalent to the target clause.

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 [Fla93].

3.6 Constrained Clauses

Constrained clauses are a special case of determinate clauses with the depth of existential quantification restricted to zero. That is, constrained clauses C contain no existential variables, more formally $vars(body(C)) \subseteq vars(head(C))$. As for determinate clauses, PI is useful to make learning succeed when the initial vocabulary was insufficient. This can be shown by the following example.

Example 7. Let $B = \phi$ and

$$\begin{array}{l} E^{\oplus} = \{\ rev([],[]),\ rev([a],[a]),\ rev([a,b],[b,a]),\ \ldots\ldots\ \} \\ E^{\ominus} = \{\ rev([a,b],[a,b]),\ rev([c],[c,e]),\ rev(a,[a]),\ \ldots\ldots\ \} \end{array}$$

Then there is no constrained solution of the learning problem that uses only $rev/2$, $[\_|\_]$ and $[]$ (though there is one with existential variables, cf. [Bun90]). However, using an additional 3-place predicate *newp* allows a constrained definition:

$$\begin{array}{ll} rev(L, LR) & \leftarrow newp(L, [], LR) \\ newp([], L, L) & \\ newp([X|R], L, L1) & \leftarrow newp(R, [X|L], L1) \end{array}$$

3.7 RUL-programs

Regular unary logic (RUL) programs [YS91] are a special case of constrained programs. They contain only unary predicates, and allow for non-variable argument terms only in the clause heads. The head arguments of clauses of the same predicate must differ in their function symbol. 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 [STW93]. In case that the example set is regular, PI is useful to make the learning task succeed.

Example 8. Let $B = \phi$,

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

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

$$\begin{array}{l} t(f(g([a|Y]))) \leftarrow q(Y). \\ q([]). \\ q([a|Y]) \leftarrow q(Y). \end{array}$$

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 might help. The algorithm described in [STW93] allows to decide whether $E^{\oplus}$ can be defined by a RUL-program at all. If yes, the necessary new predicates are introduced, otherwise the algorithm fails. As the examples for the new predicates are structurally less complex than the examples in $E^{\oplus}$, infinite loops of new predicates cannot occur.

4 Uselessness Results

In spite of the general utility of PI as bias shift operation for fixed size languages, there are language biases for which even PI fails to extend the range of expressible concepts. In particular, function-free languages exhibit this kind of weakness.

Proving the uselessness of PI according to definition 1 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.

4.1 Function-Free Constrained Clauses

Function-free constrained clauses are constrained clauses without any functors except for finitely many constants. In contrast to the general case of constrained clauses, PI is useless to recover from a failure of the learning task.

Example 9. Let examples about *grandparent*(X, Y) be given for a set of persons, and let the background knowledge contain all *parent*-relations between them. Then there is no non-trivial constrained program that covers the examples. Furthermore, PI is useless as it cannot help to introduce the necessary existential variable.

The following theorem proves this assertion.

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

Proof. We assume that a solution P' in L' exists. New predicates in P' can be eliminated without changing the success set, because recursive calls of them are applied to permutations of the head arguments. As there are only finitely many, non-recursive definitions can be determined that allow to eliminate the new predicates in P'. This results in a complete and consistent P'' in L, in contradiction to the precondition of the theorem.
Construction of P'' from P': Let

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

$$\begin{aligned}\text{Let } A_0 &= C_{newp}\\ A_{i+1} &= A_i \cup \{C \mid \exists C_1, C_2 \in A_i\ (C = (C_1 \cdot C_2\sigma)^4) \wedge \\ &\qquad \neg\exists C' \in A_i\ (C'\theta \subseteq C)\}\end{aligned}$$

As there are only finitely many clauses in each constrained function-free language L', there is an integer n such that $A_{n+1} = A_n$. Let

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

Then A has the same success set as C_{newp}, that is $A \vdash a$ iff $C_{newp} \vdash a$:

$\Rightarrow$: Let $A \vdash a$ be true. For each clause $C \in A - C_{newp}$ used in the proof, there is a resolution derivation from C_{newp}. Therefore, there is a resolution proof $C_{newp} \vdash a$.

$\Leftarrow$: 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. For each possibility to resolve them away, there is a corresponding clause in A. Therefore, there is a resolution proof $A \vdash a$.

[4] Here, $(A \cdot B\sigma)$ is the result of resolving A and B with substitution σ.

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 program P'' in L.

That is, in function-free constrained Horn logic PI is useless in case that 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. 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 5. *Given an arbitrary monadic logic program P' in a language L, there exists a program P 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. Therefore they can be eliminated from P' without changing the success set. A detailed description of the construction of P from P' can be found in [STU1].

That is, 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. Excluding arbitrary functors leads to the decidability of logical implication, in contrast to full first order Horn logic.

This decidability accounts for the uselessness of PI. In [Sta93] 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 P. The parameter n depends on the number of constants in B and L, and the arity of the available predicates. This property accounts on the one hand for the decidability of the learning problem, but on the other hand for the uselessness of PI. As only clauses with up to n different variables need to be considered, recursively defined new predicates can be eliminated by a method similar to that for constrained clauses.

Theorem 6. *If there is no function-free solution P to the learning problem $(E^{\oplus}, E^{\ominus}, B, L)$, then there is also no function-free solution P' in L' for each extension L' of L with finitely many new predicate symbols.*

Proof. As in the proof of theorem 4, we assume that a complete and consistent P' in L' exists, and construct a complete and consistent P'' in L, in contradiction to the precondition of the theorem. The crucial difference from theorem 4 is in the inductive construction of the set A of non-recursive new predicate definitions:

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

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 [Sta93], an extensionally equivalent set A_{i+1} of clauses with at most n variables is constructed from B_{i+1}. As there are only finitely many clauses 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}. As 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 P''.

That is, if the learning method fails to find a function-free solution, PI is useless to recover from the failure. However, if the validity and predictiveness of the induced program is to exceed the given constants, new predicates might be necessary for the learning task.

Example 10. Let examples about *male_ancestor*(X, Y) be given for a set of persons, and let the background knowledge contain the *parent*(X, Y)- and *male*-relations for them. Then a correct function-free target program can be found that uses only *parent* and *male* within the clause bodies. However, this program will fit exactly the family relations between the known people. For arbitrary persons, it will generally produce no predictions. Therefore, if arbitrary many persons are to be considered, only the introduction of a new recursively defined predicate as e.g. *ancestor* might help.

However, the example refers to a fundamentally 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. This violates in fact the restriction to function-free programs, as potentially infinitely many constants need to be considered.

5 Characterizing Languages with respect to Predicate Invention

In the previous sections we discussed the usefulness of PI for different language biases. The results are summarized in figure 1. It shows that the more restricted the target language is, the less useful is PI.

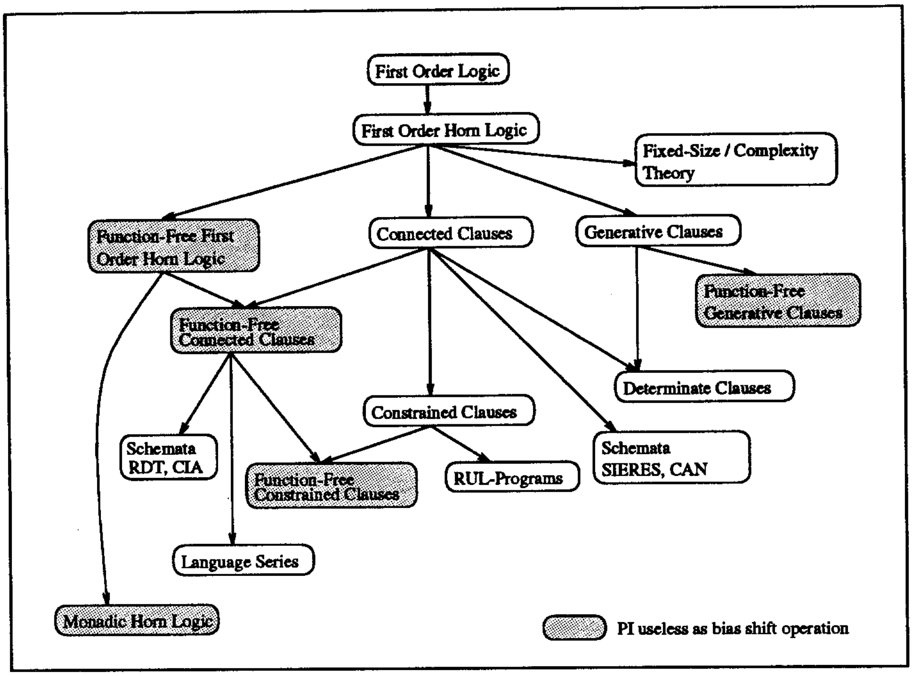

First Order Logic
First Order Horn Logic
Fixed-Size / Complexity Theory
Function-Free First Order Horn Logic
Connected Clauses
Generative Clauses
Function-Free Generative Clauses
Function-Free Connected Clauses
Determinate Clauses
Constrained Clauses
Schemata RDT, CIA
Function-Free Constrained Clauses
RUL-Programs
Schemata SIERES, CAN
Language Series
Monadic Horn Logic
PI useless as bias shift operation

Fig. 1. Usefulness of PI as bias shift operation

Especially for function-free languages, the introduction of new predicates does not increase the expressiveness of the the original language. Function-free languages are quite weak as the inductive inference method can never leave the space of what is expressible with the given constants and predicates. If the validity of the induced program is to exceed the given constants, PI might be useful. However, in that case the learning task is as difficult as in unrestricted Horn logic.

The languages for which PI is a useful bias shift operation can be distinguished in two classes. The first is the class of unrestricted or weakly restricted Horn clause languages. If learning fails in a language of that kind, PI might help to overcome the limitations of the given vocabulary. Necessary new predicates must be defined recursively in these languages, as else the original learning task would not have failed. That is, PI really has the capability to introduce new predicates missing in the original vocabulary. However, this capability comes at the price of the undecidability of the problem when to introduce a new predicate. In [STU1] we prove that it is undecidable whether a learning task $(E^{\oplus}, E^{\ominus}, B, L)$ fails in an unrestricted or weakly restricted Horn clause language L, that is whether L should be extended to make learning succeed.

The second class of languages for which PI is useful contains languages restricted to a fixed size by language parameters, schemes or size bounds. For each learning problem, these languages result in a finite hypothesis space. In this framework, new predicates mainly serve the task of extending the given fixed-size language without violating the specified parameters or schemes. They do the same job as shifts to a more general language as e.g. in CLINT [Rae92], increasing parameters as e.g. for *ij*-determinacy [MF90], or supplying more complex schemes. In the strict logical sense, most of these predicates are not necessary as they can be eliminated by unfold-operations. However, in contrast to the pure language shifts, new predicates allow additionally for expressing recursive subrelations as e.g. in example 3. Therefore, PI is a more powerful bias shift operation.

A special case of PI is the transformation of non-determinate background literals in functional form. It involves detecting dependencies between arguments of the literals, and restructuring the knowledge base accordingly. This kind of PI is employed in the context of inductive data engineering [Fla93], where a relational data base is restructured according to inductively detected attribute dependencies.

6 Conclusions

The central aim of PI in ILP is to extend the given vocabulary in case it is insufficient for the learning task. The utility of PI as bias shift operation depends on its prior utility for the current language bias. The theoretical results presented in this paper mark the boundaries of appropriatness for PI in ILP. Though they give no practical algorithms for efficiently deciding when PI is actual necessary, they indicate for which language biases it is useful at all.

Three classes of language biases can be identified with respect to the utility of PI. For function-free languages, PI can be proved as useless. These languages are restricted so strongly that PI cannot increase their expressiveness. Only allowing negation by failure leads to a useful form of PI, the so-called closed world specialisation [BM92].

For unrestricted or weakly restricted Horn clause languages, PI is useful and really capable of introducing predicates missing in the original language. However, inductive inference is unfeasible in these frameworks.

The remaining class of fixed-size languages allows for tractable induction procedures. Additionally, PI is useful both for shifting the language bias syntactically and extending the vocabulary with necessary new predicates. That is, for these languages PI integrates two different bias shift operations in one. The capabilities of PI when compared to changing the language parameters or supplying more complex schemes need to be explored further.

Furthermore, the utility of PI with respect to algorithmic biases evidenced in the work on constructive induction in propositional learning has to be studied separately. This might lead to more practical results that those in this paper.

Acknowledgements

This work has been supported by the European Community ESPRIT BRA 6020 ILP (Inductive Logic Programming).

References

[BM92] Bain, M., Muggleton, S. (1992): *Non-Monotonic Learing*in S. Muggleton (ed): Inductive Logic Programming, Academic Press

[Bun90] Buntine, W. (1990): *Constructive Induction in Definite Clause Logic*, draft

[Fla93] Flach, P. A. (1993): *Predicate Invention in Inductive Data Engineering,* Proceedings of the European Conference on Machine Learning, Vienna

[Kie91] Kietz, J., Wrobel, S. (1991): *Controlling the Complexity of Learning in Logic through Syntactic and Task-Oriented Models*, in S. Muggleton (ed): Inductive Logic Programming, Academic Press

[Kle52] Kleene, S. C. (1952): *Finite Axiomatizability of Theories in the Predicate Calculus Using Additional Predicate Symbols* in S. C. Kleene: Two Papers on the Predicate Calculus, Memoirs of the American Mathematical Society No. 10

[Lin91] Ling, C. X. (1991): *Inventing Necessary Theoretical Terms in Scientific Discovery and Inductive Logic Programming*, Report No. 302, Dept. of Computer Science, University of Western Ontario, London, Ontario

[Mug88] Muggleton, S. (1988): *A Strategy for Constructing New Predicates in First Order Logic*, in Proceedings of the Third European Working Session on Learning, Pitman

[MB88] Muggleton, S., Buntine, W. (1988): *Machine Invention of First-Order Predicates by Inverting Resolution*, Proceedings of the 5th International Conference on Machine Learning, Morgan Kaufman

[MF90] Muggleton, S., Feng, C. (1990): *Efficient Induction of Logic Programs*, Proceedings of the 1st Conference on Algorithmic Learning Theory, Tokyo, OHMSHA

[Mug92] Muggleton, S. (1992): *Inductive Logic Programming*, in S. Muggleton (ed): Inductive Logic Programming, Academic Press

[MSB92] Muggleton, S., Srinivasan, A., Bain, M. (1992): *Compression, Significance and Accuracy*, in Proceedings of the Ninth International Machine Learning Conference, Morgan Kaufmann

[Qui90] Quinlan, J. R. (1990): *Learning Logical Definitions from Relations*, Machine Learning 5

[Rae91] De Raedt, L. (1992): *Interactive Theory Revision: an Inductive Logic Programming Approach*, Academic Press

[Rae92] De Raedt, L., Bruynooghe, M. (1992): *Interactive Concept-Learning and Constructive Induction by Analogy*, Machine Learning 8(2)

[Rou91] Rouveirol, C. (1991): *ITOU: Induction of First Order Theories*, in S. Muggleton (ed): Inductive Logic Programming, Academic Press

[Sha83] Shapiro, E. Y. (1983): *Algorithmic Program Debugging*, MIT Press, Cambridge Mass.

[SMB92] Srinivasan, A., Muggleton, S., Bain, M. (1992): *Distinguishing Exceptions from Noise in Non-Monotonic Learning*, in Proceedings of ILP'92, Tokyo

[STW93] Stahl, I., Tausend, B., Wirth, R. (1993): *Two Methods for Improving Inductive Logic Programming Systems,* Proceedings of the European Conference on Machine Learning, Vienna

[STU1] Stahl, I. (1993): *Predicate Invention in ILP – Decidability, Utility and Decision Criteria* , Deliverable STU1 of the ESPRIT BRA 6020 ILP, September 1993

[Sta93] Stahl, I. (1993): *Properties of Inductive Logic Programming in Function-Free Horn Logic*, this volume

[Tau92] Tausend, B. (1992): *Using and Adapting Schemes for the Induction of Horn Clauses*, ECAI-92 Workshop on Logical Approaches to Machine Learning, Vienna

[Wir89] Wirth, R. (1989): *Lernverfahren zur Vervollständigung von Hornklauselmengen durch inverse Resolution*, Dissertation, Universität Stuttgart, Institut für Informatik

[Wir91a] Wirth, R., O'Rorke, P. (1991): *Constraints on Predicate Invention* in Proceedings of the Eighth International Workshop on Machine Learning, Morgan Kaufmann

[Wro] Wrobel, S.: *Exploiting a Problem-Solving Context to Focus Concept Formation*, to appear in Machine Learning Journal

[YS91] Yardeni, E., Shapiro, E. (1991): A Type System for Logic Programs, *Journal of Logic Programming 10.*