Declarative Bias in ILP

Claire Nédellec, Céline Rouveirol
Laboratoire de Recherche en Informatique,
Inference and Learning Group,
Université Paris Sud, bât 490
F-91405 Orsay, France

Hilde Adé
Department of Computer Science
Celestijenlaan 200A
B-3001 Heverlee, Belgium

Francesco Bergadano
Università di Torino
Dipartimento di Informatica
Corso Svizzera 185
10149 Torino, Italy

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

Abstract. Interest in Declarative bias in Machine Learning is growing with the expressivity of the concept description language of ML systems. Inductive Logic Programming more than any other ML field is thus concerned with explicitely biasing learning. The main issues already identified in declarative bias [RG90] have been studied within the ILP project, i.e. the restriction of the size of the search space for the target concept and representation of the bias. As a first step, an extensive study of existing ILP systems and the elicitation of the role of hidden bias has led to define typologies of bias in relation with their effects on the learning process as well as alternative representation for bias. Declarative representations of bias have been defined through different types of languages so that bias can be easily set and shifted. In parallel with the definition, the representation and the experimentation of various biases, the interactions between different types of bias have been analyzed in term of computational learning cost so that reusable guidelines for bias setting may be provided.

1 Introduction

1.1 Motivations

The issues on declarative bias studied within the ILP project, i.e. the restriction of the size of the search space for the target concept, the control of its exploration, and the declarative representation of bias, are motivated by the following results in ML.

The importance of bias was experimented with in early Machine Learning and Pattern Recognition research [Cov69]. In particular, it was observed that stronger language bias (i.e., smaller hypothesis spaces) would lead to better performance of the learned programs on a test set of new examples not available to the system during the learning phase. This fact has been explained in beautiful generality by the works of Vapnik and Chervonenkis [VC81] and further developed within the Computational Learning

Theory community and in the Pattern Recognition literature [Dev88, SB93]. The first observation of Vapnik and Chervonenkis is based directly on Bernoulli's Theorem: the difference between the percentage of errors made by the learned program on the given examples and the same figure measured on a separate set of test data grows with the size of the hypothesis space $\mathcal{P}$ of possible programs. The main result of Vapnik and Chervonenkis is that the same holds for infinite hypothesis spaces, where cardinality is replaced by a measure of the expressiveness of the hypothesis space, the so-called Vapnik-Chervonenkis dimension of $\mathcal{P}$.

The hypothesis language used in ILP is often Horn clauses: by far a more expressive language than those used in Pattern Recognition (e.g., linear discriminants or prototypes for nearest neighbor classification) and in propositional Machine Learning (e.g., decision trees or propositional classifiers). For the concerns on accuracy, Horn clauses are hopeless: even the weakest notions of inductive success, such as identification in the limit cannot be met without appropriate restrictions. Some form of language bias is therefore necessary for restricting the hypothesis space, more necessary than it had been in previous Machine Learning research. The hypothesis spaces of possible logic programs must be extremely restricted, based on specific knowledge about the problem at hand, for otherwise the behavior of the learned program on new data can be unpredictable, and unrelated to completeness and consistency as measured on the training examples.

Obviously, the hypothesis space cannot be too small either, because it must contain some program that has an acceptable accuracy on the available data. Bias must, then, be weak enough to allow for complete and consistent programs, and strong enough to prevent total degradation of performance on new data.

Bias is not limited to the definition of a hypothesis space, and is also central to addressing efficiency concerns, besides the issue of accuracy. The biased hypothesis space can be wide enough so that extensively searching it may be unrrealistic from a computational point of view. [Mit80] described learning in term of problem solving where bias are considered as heuristics that efficiently guide the search through the hypothesis space. In this framework, various types of bias have been studied such as preference criteria, search biases and stopping criteria that exploit the available classified examples and order relation on the hypothesis space in order to avoid an extensive exploration of the hypothesis space.

The importance of a *declarative* description of bias has been stressed by [RG90] and [RG90] among others. They argue that a declarative description of the language bias is a condition sine qua non for the automation of language bias selection and shift of language bias. More recently, [DB92] showed that expressing declaratively learning strategies in generic algorithms provides a general framework, both for comparing and adapting ML methods.

1.2 Declarative bias in ILP

The declarative representation of bias and its operationalisation in implemented systems requires the clarification of the notion of bias, that is, clearly defining the basic knowledge and the learning operations, and the biases that affect them. Based on the identified biases, section 2 intoduces typologies of bias. In section 3, representations of bias are defined in relation to the learning task to achieve. Three complementary types of language of bias for language bias setting, *Clause sets*, *MILES-CTL* and $\mathcal{D}$LAB have been developed for different purposes (section 3.2). Configuration of generic ML methods is an alternative solution for bias representation and setting that has been explored with the knowledge level description and implementation of a generic generate-and-test strategy in the system $\mathcal{HAIKU}$ (section 3.3). Some properties of the association of language and search bias have been identified based on extensive experimentation with $\mathcal{HAIKU}$(section 4.3). The formalization of these properties has led to define general guidelines for a more efficient implementation of biases.

2 Definitions and typologies of bias

The following typologies of bias (sections 2.2 and 2.3) we have defined, are based on previous general definitions of bias, that appeared as too general to be fully operational in ILP, but formed an interesting basis for discussion.

2.1 Definitions

Definition 1 *(after [Mit80]) We use the term bias to refer to any basis for choosing one generalization over another, other than strict consistency with the observed training examples.*

Definition 2 *(after [Utg86]) Except for the presented examples and counterexamples of the concept being learned, all factors that influence hypothesis selection constitute bias. These factors include the following:*

- *The language in which hypotheses are described.*
- *The space of hypotheses that the program can consider.*
- *The procedures that define in what order hypotheses are to be considered.*
- *The acceptance criteria that define whether a search procedure may stop with a given hypothesis or should continue searching for a better choice.*

2.2 A typology of bias

There is now some agreement on dividing the notion of inductive bias based on Utgoff's definition into three different categories, namely, *language bias*, *search bias*, and *validation bias.*

- The *language bias* defines the target concept language, by determining the hypothesis space of the possible concept descriptions. It restricts the *Concept description language.* that hold for all concept learning. The *language bias* includes any language restriction that is specific to the current concept to learn. For instance, the concept description language may be Horn clauses without function symbols and the language bias may restrict the target concept language to clauses with only 4 literals in the body.

- The *search bias* determines which part of the hypothesis space is searched, and how it is searched. It can be a *restriction* or a *preference* bias. A restriction bias determines which programs should be ignored, while a preference bias determines which program should be considered first, which clauses should be considered first, and which predicates should be first added or removed from clause antecedents. This is a generalization of what had been called a preference criterion [Mic83b].

- The *validation bias* [Mic83b] determines an acceptance criterion for the learning system, telling the system when the search for the desired logic program should stop. This could happen whenever the learned program is complete and consistent with respect to the given examples. In most Pattern Recognition applications, one would rather stop when some *degree* of completeness and consistency has been reached. Such a choice has also been made in some ILP applications [KMS92].

2.3 Refinement of the general typology

To reflect the complexity of the exploration of the search space by various ILP methods, [NR94] has refined the above notions of search bias and validation bias in the following way.

2.3.1 Search bias

Search bias can be decomposed into the following parameters.

- *Ordering relation on the hypothesis space.* In a generative approach, where the candidate hypotheses are generated by altering a current concept definition into a "better one", the notion of search bias includes the operator that alters the current concept definition, such as for instance, inversion of resolution, dropping-literal, etc. It may be based on any partial order relation on the hypotheses such as any generality relation (see [NR94]) that allows to prune the inconsistent hypotheses from the hypothesis space. The generality relation is classically based on background knowledge, that then plays the role of a bias. In a static approach (as opposed to generative), the notion of search bias includes the order in which the pre-compiled hypotheses are to be tested against the examples.

- *Example selection criteria.* The way examples are selected among the available ones by the ML systems or by the user is also a bias: a given candidate hypothesis is considered as valid or not depending to its consistency and its completeness with respect to the *selected* examples. This holds whether the selected example is used to drive learning as in a data-driven strategy or the selected example is used to validate the current candidate hypothesis in a generate-and-test strategy.

- *Coverage function.* It determines if a given example is covered or not by a given hypothesis. It is a bias in the sense that it is used for checking the correctness the hypotheses. It may be based on resolution or θ-subsumption and it uses or not the background knowledge that once again can be exploited as a search bias.

- *Intermediate validation criteria.* This criterion determines the validation of a candidate hypothesis with respect to the examples. Classically, the criterion tests the consistency of the hypotheses with respect to the examples selected by the example selection criteria. It may be absolute: the hypothesis has to be consistent with *all* the selected examples, or relative: the hypothesis has to be consistent with a *subset* of the selected examples such as the discrimination measures used in FOIL. A given hypothesis may considered as "consistent" for a given intermediate validation criteria while it is not for another. In this sense, it influences the search through the hypothesis space and the learning result.

2.3.2 Validation bias

[NR94] distinguishes two kinds of validation bias, validation of a partial concept definition or program and stopping criterion.

- *Validation criterion of a partial concept definition.* In case of multi-concept learning or "disjunctive" concept description language, the learning algorithm requires an additional criterion to determine if a given clause is part of the target concept definition or not. In most systems, it is relative consistancy and measured according to a threshold, or absolute consistency with respect to the whole set of examples.

- *Stopping criterion.* It determines when learning must stop. It is usually relative (to a threshold) or absolute completeness with respect to the whole set of examples.

Single concept learning in conjunctive concept description language obviously requires a single criteria.

2.4 Declarativity

Eliciting biases and defining a typology that clearly states their roles is a first step towards designing systems that can be adapted to the learning task. But to achieve this goal, a *declarative representation of the bias* is also required so that bias setting and shifting can be easily performed. Here, bias is said as being declarative once it is explicitely represented, - beit at the knowledge level, or in the system specification - and not hardwired in the system's implementation. Bias declaration must be such that one can reason about it, i.e., such that the correlation between input and output of a learning system can be clearly defined. Within this framework, the role of background knowledge as declarative bias is subject to discussion as it clearly depends on the way it is exploited. Background knowledge is traditionally viewed as one input of inductive learners, i.e., it has the same status as the examples. This is clearly contradictory with Mitchell's definition (section 2.1). One can argue that this is only due to historical reasons as the background knowledge has not been exploited in the former systems as it is in ILP systems.

The declarativity of the representation of the Background Knowledge (*BK*) obviously depends on the way it is exploited and the role it plays in learning. It may be act as language bias as well as search bias.

- *Background Knowledge as language bias.* When *BK* is available, it usually determines the vocabulary of the concept definition language. For instance, in clause models approach (section 3.2.1), the predicates used in the clauses of the hypotheses are determined by the clause sets, which in turn are defined in terms of the predicates in the background knowledge and predicates for which clauses have to be constructed. This means that the background knowledge determines the predicates that are in the concept definition language. In MILES-CTL (section 3.2.3), the predicates in the vocabulary of the hypothesis language can be defined as some subset of predicates in the background knowledge. In case the search operator makes use of *BK*, BK also determines the syntax of the concept definition language as in $\mathcal{HAIKU}$ (section 3.3).

- *Background Knowledge as search bias.* Background Knowledge may influence the search in three ways. It may be used to define a generality relation on the hypothesis space and then a partial order for search. It may also be used to define a coverage relation between the examples and the hypotheses and it then determines whether a hypothesis is correct and complete or not. It may also be used for checking the semantic criteria of the language (e.g. the determinacy restriction and input-output modes) in relation with the coverage relation.

  As these generality relation, coverage relation and semantic criteria are not *explicitely* expressed from the background knowledge, but have to be computed by non trivial methods, the status of BK as declarative search bias is an open question.

3 Declarative representation of bias

The declarative representation of bias may be achieved in two ways, by *languages* that allow bias specification, or by *configurable generic methods* that allow both specification and implementation of bias. Both methods will be presented in the following through examples.

3.1 Language bias

The declarative representations described below are provided with a number of options corresponding to general constraints on the language, some of which have been used before in ILP (see [BG95])

1. *Predicate modes* may be provided that indicate their desired input/output behavior. Every argument of a predicate is labeled as either input or output. Modes need not be unique, e.g., we may have both append(input, input, output) and append(output, output, input). More sophisticated mode declarations involving determinate and nondeterminate literals are found in the PROGOL system [SMK94]. Clause Sets can be reduced according to predicate modes as follows: a clause is permitted only if any input variable of any predicate in the antecedent either occurs earlier in the antecedent or is an input variable in the head. This is called a requirement of instantiated inputs.

2. The *output variables in the head* of a possible clause may be required to occur as an output of some predicate in the antecedent. This guarantees that an instantiated output is produced when the clause is called.

3. Once an output is produced, one may require that it is *not instantiated again*. Syntactically, this means that any variable cannot occur as output in the antecedent more than once.

4. *Forcing outputs to be used*: any output variable in the antecedent must occur either as input in some literal to the right or as output in the head.

5. *Forcing inputs to be used*: all input variables in the head must occur in the antecedent.

6. *Forbidden clauses*: it is sometimes easier to define a large set of clauses, and list a smaller number of clauses to be removed from the set, rather to list one by one the clauses that are possible. The procedure for generating a set of possible clauses also accepts in input a set of forbidden clauses.

7. *Forbidden conjunctions*: with the same motivations as above, some conjunctions of literals may be ruled out, meaning that no antecedent of a possible clause may contain them. This can be useful for two reasons: the conjunction is always true, e.g., Y is X+1 $\wedge$ X=Y-1, or it is contradictory, e.g., head(L,A) $\wedge$ null(L).

3.2 Language of bias

Three complementary languages of bias have been defined within the ILP project that allow a user to easily specify and shift different type of language biases. The choice of a language among the possible ones will depend on the characteristics of the learning task to achieve, i.e. the level of expressiveness and comprehensibility of the target description language.

3.2.1 Clause Set Formalism

Clause sets allows to specify bias that adopts standard Prolog notation, with the addition of clause, literal, and term sets.

Clause sets:

The hypothesis space will be described by a pair $< knownclauses, possibleclauses >$. A program P belongs to the hypothesis space if $P = knownclauses \cup P1$, where $P1$ is a subset of the possible clauses. The simplest syntactic tool for specifying such inductive bias is given by *Clause Sets*: known clauses are listed as in a Prolog program, while possible clauses are surrounded by brackets denoting a set.

Clause Set language has been implemented so that the systems FILP and TRACY can learn acceptable programs among the possible ones generated from the actual specifications of clause sets.

For instance, there follows a possible description of a priori information for learning a logic program for *member*:

$$\begin{array}{ll} member(X,[X|\_]). & \{member(X,[Y|Z]) :- X \neq Y, member(X,Z). \\ \{member(X,[Y|Z]) :- cons(X,W,Z).\} & member(X,[Y|Z]) :- cons(Y,Z,W). \\ cons(X,Y,[X|Y]). & member(X,[Y|Z]) :- member(X,Z).\} \end{array}$$

This means that the learned program will have to include the first clause, which is known, possibly followed by the second clause $member(X, [Y|Z]) : -cons(X, W, Z).$; the third clause $cons(X, Y, [X|Y])$. will have to follow. Finally, some or all of the remaining clauses may be appended to the program. There are sixteen different logic programs satisfying these very strict requirements; among these some represent a correct implementation of *member*. All the user will need to do at this point is provide positive and negative examples, e.g., $member(a, [b, a]), member(c, [b, d, c]) and \neg member(a, [b])$.

The learning systems FILP and TRACY could select, from among the 16 possible inductive hypotheses, a program deriving the positive examples and not deriving any negative example. As the bias is so strong, the task is very easy in this case, and the learned program can only be a correct version of *member*.

Literal sets: Unfortunately, a priori information is not always so precise, and the set of possible clauses may be much larger. As a consequence, the user may find it awkward, or even impossible, to type them one after the other. To this purpose we define *literal sets*. If a clause occurs in a clause set, then some conjunctions of literals of its antecedent may be surrounded by brackets, denoting a set. In this case the clause represents an *expansion* given by all the clauses that may be obtained by deleting one or more literals from the set. Formally:

$$\{P \text{ :- } A,\ \{B,\ C,\ ...\},\ D\} = \{P \text{ :- } A,\ \{C,\ ...\},\ D\} \cup \{P \text{ :- } A,\ B,\ \{C,\ ...\},\ D\}$$

A case with variables is shown in figure 1.

```
q(X) :- b(X).
{p(X) :- {q(X), a(X,Y)}.
q(X) :- c(X,Z), {q(Z)}}.
```

(a)

```
q(X) :- b(X).
{p(X).
p(X) :- a(X,Y).
p(X) :- q(X).
p(X) :- q(X), a(X,Y).
q(X) :- c(X,Z).
q(X) :- c(X,Z), q(Z).}
```

(b)

Figure 1: (a) Clauses with literal sets ... (b) and their expansion

In other words, the expansion of a clause is the set of clauses obtained by replacing the literal sets with a conjunction of any of its literals. With this syntactic mechanism, one can define in a concise way a large set of possible clauses.

Term sets: For defining sets of possible clauses even more concisely, our last tool is given by *term sets*: a term occurring in a literal within a literal set may be replaced by a set of terms listed within brackets. The literal is then duplicated in the set where it occurs with different arguments, as indicated by the term set. For instance:

$\{...,\ p(\{X,Y,Z\},W),\ ...\}$ is the same as: $\{...,\ p(X,W),\ p(Y,W),\ p(Z,W),\ ...\}$.

In general, term sets are expanded by means of the following rule:

$$\{p(..., \{t_1, ..., t_n\}, ...)\} = \{p(..., t_1, ...), ..., p(..., t_n, ...)\}. \tag{1}$$

This syntactic device is useful to avoid the rewriting of a predicate with similar arguments. The a priori information shown in figure 2 defines a larger set of $2^{10} + 2^5 + 2^9 = 1568$ possible clauses by using term sets. Inductive bias specified as described above is both flexible and adequate for the inductive synthesis of more complex programs. Flexibility is achieved by allowing the user to provide a priori information of different strength and to adapt it to the particular case study. Stronger and more informative prior knowledge is defined by a large set of known clauses and a limited number of possible clauses, i.e., the literal sets must be small and well constrained within the structure of the rest of the program. This will lead to efficient induction, and few examples are required.

```
{
int(X,Y,Z):-{null({X,Z}), head(X,X1), tail(X,X2),
             assign(W,Z), cons(X1,W,Z),
             member(X1,Y), notmember(X1,Y),
             int(X2,Y,{Z,W})
             }.
member(X,Y):-{head(Y,{X,KW}), tail(Y,Tail),
              member({X,KW},Tail)}.
notmember(X,Y) :- {head({Y,Z},{X,KW}), diff(X,KW),
                   tail({Y,Z},Tail), notmember({X,KW},Tail)}.
}
notmember(X,[]).
null([]).
cons(X,Y,[X|Y]).
head(X,[X|_]).
tail(X,[_|X]).
diff(X,Y) :- X≠Y.
assign(X,X).
```

Figure 2: Inductive bias for *intersection*, with term sets

If such strong information is not available, while examples abound, then a larger set of possible clauses may be defined with more frequent literal and term sets.

The Clause Set language is also provided with general mechanisms for eliminating clauses from a set that is considered too large. In particular, the procedure that translates the clause set formalism into a set of possible clauses accepts all the language constraints defined in section 3.1. All of these restrictions are syntactic and can be determined before looking at the examples. The preprocessor that generates a file of possible clauses, given a clause set specification of the bias, will check whether the language constraints are satisfied, depending on the selected options. This check is performed *during* the generation, and therefore the clauses that should not be included are never added to the list.

As a conclusion, Clause Sets approach can not only be applied to simple programs but also to complex, real-size programs, and represents a new perspective on inductive synthesis. Learning very complex programs is probably just a dream, but this does not mean that ILP cannot be a useful tool for building large software systems. Most clauses in these programs must be hand-coded, and the basic structure of the program must be determined with traditional techniques. However, Clause Sets approach would allow the programmer to leave some parts of the code underdetermined, and to fill the gaps with examples. Inductive bias could be defined in this framework and would look like a normal Prolog program, with small literal sets and optional clauses occurring here and there.

3.2.2 *Effects on language bias on Hypothesis Space size*

The effects on the hypothesis space of the language bias that can be specified by Clause Set language and have been experimentally measured independently of any search bias. This has been done by configuring the generic system $\mathcal{HAIKU}$ described below (section 3.3) and experimenting with the bias to be tested. As a first step, the effect of the following language biases on the size of the hypotheses space have been tested. The items the following language biases referred to, are those defined in section 3.1.

1. *Number of literals in the body of the clause (NL),*
2. *Number of variables (NV),*
3. *Forbidden literals (from item 6.), (FL),*
4. *Integrity constraint (from item 7. forbidden conjunctions, (IC),*
5. *Redundancy (restrict the generation of equivalent clauses), (RE),*
6. *Range-restriction (RR),*
7. *Connexion (CO),*
8. *Instantiation of the input variables (from item 1.), (II),*
9. *Head output must appear in the body (from item 2.), (IO),*
10. *Overloaded output (from item 3.), (OO),*
11. *Use of the body output variables (from item 4.), (UO),*
12. *Head input must occur in the body (from item 5.), (UI).*

Items 1 and 2 directly derive from clause sets specification. Items 4 and 5 represent biases that may be based on background knowledge. Items 8 to 12 define bias based on input-output modes. One can find a detailed report of the way these biases have been defined in $\mathcal{HAIKU}$ and how the results may be interpreted in [LRI95]. As a summary, it appears in toy examples studied that each of those biases are either very restricting or very weak. We will present here the results obtained on the intersection example only, more details may be found in [LRI95].

Measures on the intersection domain

This example from [Ber93] allows the study of the effects of language bias based on functionality.

The concept *null*, *head*, *tail*, *member* et *notmember* are pre-defined with the modes: *intersection(in,in,out)*, *null(out)*, *head(in,out)*, *tail(in,out)*, *member(in,in)*, *notmember(in,in)*
and the following program has to be learned,

$$
\begin{aligned}
intersection(X,Y,Z) &\leftarrow null(X), null(Z).\\
intersection(X,Y,Z) &\leftarrow head(X,HX), tail(X,TX), member(HX,Y),\\
&\qquad intersection(TX,Y,W), cons(HX,W,Z).\\
intersection(X,Y,Z) &\leftarrow head(X,HX), tail(X,TX), notmember(HX,Y),\\
&\qquad intersection(TX,Y,Z).
\end{aligned}
$$

The concept description language contains all predicates and is restricted to the clauses that verify the appropriate type (e.g. the arguments of the predicates are specified as lists or elements of lists.) Literals in the form of *intersection (A,A,_)* and *cons(_,A,A)* are forbidden. Integrity constraints avoid an empty list to be tested and that a given element both belong and does not belong to a list.

NL	: Nb of literals restricted to 5	**NV**	: Nb of existential variables restricted to 3
CO	: Connexion	**IO**	: Head *output* in the body
UO	: Use of the *output*	**OO**	: No overloaded *output*
II	: Instantiation of the *input*	**FL**	: Forbidden literals
IC	: Integrity constraints		

NL	NV	CO	IO	UO	OO	II	FL	IC	HS size	%
									2^{101}	100
X									83 463 472	10^{-21}
	X								3×10^{11}	10^{-17}
		X							*	*
			X						$2^{101} - 2^{73}$	99.9
				X					*	*
					X				435×2^{45}	10^{-13}
						X			6×10^{26}	0.024
							X		2^{96}	3.125
								X	2197×2^{56}	10^{-9}

* : not performed computation

Table 1: Measures on the intersection example

The results of the experimentation are given in table 1.

In this domain and the domains presented in [Tor95], the effects of language biases appeared as very extreme. [Tor95] suggest some strategy to exploit this observation.

3.2.3 A scheme based language for language bias

The basic idea of the representation MILES-CTL [Tau94a] is an extension of the approach in MOBAL [MWKE94], i.e., sets of hypotheses are described by schemes. The aim for developping MILES-CTL is to achieve an empirical comparison of the biases used in ILP in terms of the size of the hypothesis space. One major prerequisite was to extend the model of inductive learning in order to reveal the different biases (section 2). They have been implemented in MILES-CTL in order to enable a systematic investigation of the language bias in ILP systems. Basic constituents of language biases have been identified, and biases have been classified with respect to the aspect of Horn clauses they influence ([Tau94b], [Tau95a]).

A scheme for a hypothesis clause called *clause template* includes schemes for each literal in the clause. Similar to record data types, a literal template consists of several identifiers followed by a scheme variable or a constant. In addition, the domain of a scheme variable in a literal template can be further restricted by conditions. Since the items in a literal template describe the set of covered literals, they have to include at least an item for the predicate name and the arguments. Other items, for example, describe the arity, the number of new variables, the argument or the predicate types or the depth of the covered literals. A declaration of a hypothesis language in MILES-CTL consists of a set of schemes T, the vocabulary Σ including predicates, functors, and types, and an instantiation function I. Given a set T and Σ, I constructs hypotheses by instantiating the scheme variables in a clause template.

For example, suppose that we are to define a hypothesis language L_{H_1} in MILES-CTL. The vocabulary Σ_1 of L_{H_1} includes predicates of type *comp* and *listp* and arity less than 4 and the instantiation function is I_1. The following sets of hypothesis clauses has to be in L_{H_1}:

- clauses with exactly one body literal the arity of which is less than 3
- clauses with two body literals where the arity of the first one is less than 2 and the predicate type of the second one is *comp*,
- clauses with two body literals where the arity of the first one is less than 2 and of the second is 3.

These clauses have to be covered by the clause templates in T_1 of L_{H_1}.
In MILES-CTL, T_1 can be defined by $T_1 = \{T1, T2, T3\}$ with

$$T1: \begin{bmatrix} \mathit{predicate} : P11, \\ \mathit{arguments} : A11 \end{bmatrix} \leftarrow \begin{bmatrix} \mathit{predicate} : P12, \\ \mathit{arguments} : A12, \\ \mathit{arity} : Ar12 \| (Ar12 \leq 3) \end{bmatrix}.$$

$$T2: \begin{bmatrix} \mathit{predicate} : P21, \\ \mathit{arguments} : A21 \end{bmatrix} \leftarrow \begin{bmatrix} \mathit{predicate} : P22, \\ \mathit{arguments} : A22, \\ \mathit{arity} : Ar22 \| (Ar22 \leq 2) \end{bmatrix}, \begin{bmatrix} \mathit{predicate} : P23, \\ \mathit{arguments} : A23, \\ \mathit{predicate\_type} : \mathit{comp} \end{bmatrix}.$$

$$T3: \begin{bmatrix} \mathit{predicate} : P31, \\ \mathit{arguments} : A31 \end{bmatrix} \leftarrow \begin{bmatrix} \mathit{predicate} : P32, \\ \mathit{arguments} : A32, \\ \mathit{arity} : Ar32 \| (Ar32 \leq 2) \end{bmatrix}, \begin{bmatrix} \mathit{predicate} : P33, \\ \mathit{arguments} : A33, \\ \mathit{arity} : 3 \end{bmatrix}.$$

In these clause templates, $P11, P23, A11, \ldots$ are scheme variables to be instantiated by I. Obviously, the first set of clauses in the specification of L_{H_1} is covered by $T1$, the second by $T2$ and the third by $T3$. This example shows how a hypothesis language can be defined declaratively in MILES-CTL language.

3.2.4 $\mathcal{D}$LAB *grammar*

$\mathcal{D}$LAB is a grammar formalism developed for representing language bias that allows for an intensional syntactic definition of the hypothesis language $\mathcal{L}$. $\mathcal{D}$LAB stands for $\mathcal{D}$eclarative LAnguage Bias. Together with the definition of $\mathcal{D}$LAB, a user-friendly tool has been developped that enables the specification of syntactic language bias. At the same time a mechanism is provided that, based on the language specification, systematically (and non-redundantly) generates all clauses in the hypothesis space.

$\mathcal{D}$LAB further extends the formalism described in [ADRB95], and combines the rule schemata of [EHR83] and the language representation of [Ber93]. The following description is extracted from [DR95] on $\mathcal{C}$lausal $\mathcal{D}$iscovery.

- *Basic* $\mathcal{D}$LAB A $\mathcal{D}$LAB grammar basically consists of a set of templates to which the clauses in the language $\mathcal{L}$ conform. Each such template has the form $DTEMP = HeadTemplate \leftarrow BodyTemplate$, where both $HeadTemplate$ and $BodyTemplate$ are $\mathcal{D}$LAB terms. Finally, a $\mathcal{D}$LAB term is either an atomic formula, or a formula of the form $Min - Max : L$, where Min and Max are integers with $0 \leq Min \leq Max \leq length(L)$, $Max > 0$, and L is a list of $\mathcal{D}$LAB terms.

  The generation of a language $\mathcal{L}$ given a $\mathcal{D}$LAB grammar then basically consists of the (recursive) selection of all subsets of L with length within range $Min \ldots Max$ from each $\mathcal{D}$LAB term $Min - Max : L$ in the grammar.

  The following example illustrates how using a $\mathcal{D}$LAB grammar, the language $\mathcal{L}$ can be generated. It gives a first idea of the expressive power of the relatively simple $\mathcal{D}$LAB formalism.

Example 1 *Given a well-formed* $\mathcal{D}$LAB *term* $Min - Max : L$*, we can distinguish the following cases of special interest:*

- **all subsets:** $Min = 0, Max = length(L)$

$$DGRAM_1 = \{0-1 : [human(X)] \leftarrow 0-2 : [female(X), male(X)]\}$$

$$\mathcal{L}_1 = \begin{cases} \leftarrow \\ human(X) \leftarrow \\ \leftarrow female(X) \\ \leftarrow male(X) \\ human(X) \leftarrow female(X) \\ human(X) \leftarrow male(X) \\ human(X) \leftarrow female(X) \wedge male(X) \end{cases}$$

- **all non-empty subsets:** $Min = 1, Max = length(L)$

$$DGRAM_2 = \{0-1 : [human(X)] \leftarrow 1-2 : [female(X), male(X)]\}$$

$$\mathcal{L}_2 = \begin{cases} \leftarrow female(X) \\ \leftarrow male(X) \\ human(X) \leftarrow female(X) \\ human(X) \leftarrow male(X) \\ human(X) \leftarrow female(X) \wedge male(X) \end{cases}$$

- **exclusive or:** $Min = Max = 1$

$$DGRAM_3 = \{0-1 : [human(X)] \leftarrow 1-1 : [female(X), male(X)]\}$$

$$\mathcal{L}_3 = \begin{cases} \leftarrow female(X) \\ \leftarrow male(X) \\ human(X) \leftarrow female(X) \\ human(X) \leftarrow male(X) \end{cases}$$

- **combined occurence:** $Min = Max = length(L)$

$$\begin{array}{lll} DGRAM_4 = & \{ & 1-1 : \ [human(X)] \leftarrow \\ & & 0-2 : \ [\\ & & \qquad 2-2 : [female(X), is\_daughter(X)], \\ & & \qquad 2-2 : [male(X), is\_son(X)] \\ & & \qquad] \\ & \} & \end{array}$$

$$\mathcal{L}_4 = \begin{cases} human(X) \leftarrow \\ human(X) \leftarrow \ female(X) \wedge is\_daughter(X) \\ human(X) \leftarrow \ male(X) \wedge is\_son(X) \\ human(X) \leftarrow \ female(X) \wedge is\_daughter(X) \wedge \\ \qquad\qquad\quad male(X) \wedge is\_son(X) \end{cases}$$

A description of an extended version of $\mathcal{D}$LAB and its application to mesh design may be found in [DR95].

3.3 Bias configuration

The languages of bias described above allow a user to easily specify and shift the concept description language, he wants a learning system to restrict its search within.

The configuration approach adopted with the $\mathcal{HAIKU}$ system differs in that the goal is to provide the user with a generic learning system that allows not only to represent but also to implement the types of bias identified in the typology (section 2).

The main results achieved consist in a complete model and implementation of $\mathcal{HAIKU}$ for the combination and the application of bias with a generate-and-test strategy. At this point, the configuration is graphically specified by the user through a panel instead of a language. In $\mathcal{HAIKU}$, the language bias is expressed in the form of criteria such as those described in section 3.1. Conjunctive and disjunctive languages are possible both for concept description language and hypothesis language.

3.3.1 Search bias in $\mathcal{HAIKU}$

The notion of declarativity of bias is usually related to the notion of shift of bias and then of genericity of the ML algorithm. It seems that the classical trade-off between efficiency and generality [LB85] also applies here. That is, the more generic a system is, the less efficient its implementation is. The gain in flexibility and adaptability by the use of bias may be partially lost in term of complexity as generic implementation requires many additional tests that are not necessary in a library of systems.

One of the main ideas that underlies the $\mathcal{HAIKU}$ system is that the effect of search bias setting w.r.t the computational cost reduction could be greatly enhanced if the architecture of the learning system itself would reflect the search method. More precisely, the computational cost of exploring a wide search space is drastically reduced if the combination of different biases is performed through the combination of pieces of software, instead of parameters to be tested at each step.

More than only enhancing the efficiency of the search, this kind of approach offer a better comprehensibility of the learning process. The generic system is associated to a model that closely reflects the learning methods implemented, and the role of its input and output. In this framework, bias appears as control knowledge the effect of which is clearly elicited in the model. Therefore, bias and its effects on the learning method are declaratively represented at both levels, model and implementation.

The fundamental strategy in $\mathcal{HAIKU}$ is based on generate-and-test method as opposed to data-driven. The detailed description of the generic algorithm may be found in [NR94]. In order to avoid an exhaustive search of the space, the hypothesis space is ordered by a generality relation (i.e, θ-subsumption, resolution, or generalized subsumption), to be set by the user as a bias. Classically, any hypothesis that is more general than an inconsistent one is pruned. The search process starts with an initial bound (upper or lower) of the hypothesis space that is built from a completion bias and optionally a single positive example. The candidate hypotheses are generated step by step from the initial bound by applying an operator in a bottom-up or top-down strategy (e.g. resolution / inversion of resolution, dropping / adding literals). The availability of completion biases and the learning operators obviously depends on whether background knowledge has been provided or not. In any case, background knowledge is reusable for a whole application.

Section 4.3 details how language bias interferes with search bias in $\mathcal{HAIKU}$ so that the search may be optimal.

3.4 Comparison of bias representation

Bias representation can be compared along two lines: the family of language it belongs to (schemes, sets or grammar) on the one hand and on the other hand, the way it is interpreted by the learning system.

3.4.1 Comparison of language of bias

Languages of bias fall into one of the following categories: the representation by parameterized languages, schemes, clause sets, grammars, or combined approaches that exploit some of the previous approaches.

- *Representation by parameterized languages*: (example, CLINT [DR92]). This type of representation of declarative bias is the first one that have been defined and implemented in ILP systems. Abstraction is achieved by describing the bias the hypothesis clauses have to verify, such as for instance, the number of existential variables, or the connexion criteria. This type of representation is not directly related to the form of the possible clauses as it is in scheme-based or Clause Sets representation.

- *Scheme-based Representations*: The scheme-based representations are graph-based or non graph-based. Graph-based approaches include, dependency graphs as in SIERES [WO91], in [Tau92], Algorithm Sketches [BJ93]. Non-graph-based include rule models [KW92], clause templates [Tau94b], relational cliches [SP91] As shown in section 3.2.3, abstraction is achieved by replacing some parts of the hypothesis clauses by variable parts, e.g. predicate name, arguments. They represent some abstraction of sets of Horn clauses. It is particularly suitable for representing hypothesis languages that consists of small subsets of possible clauses with different features.

- *Representation by clause sets*: (see section 3.2.1) This representation overcomes one of the limitation of the scheme-based representation by enabling a concise expression of variable numbers of clauses, literals and terms. However the representation of language biases that are not related to clause, term or predicate sets, such as mode declarations, or argument types, are not as integrated in this representation as in scheme-based approaches.

- *Representation by grammars*: (for example, GRENDEL [Coh93]) Abstraction is achieved by grammar rules the application of which can be restricted by various types of biases. Large languages can be very concisely represented. All types of biases can be easily integrated in grammar specification but as the representation is not close to the target representation, i.e., programs, the concept definition language generated is not as easily predictable for a "naive" user, as it may be in scheme-based or Clause Set approaches. Moreover, the order in which clauses are generated cannot be controlled as easily.

- *Combined approaches*: GRDT [Kli94] and $\mathcal{D}$LAB grammar as shown in section 3.2.4, combine the rule schemata of [EHR83] and Clause Set language.

3.4.2 Comparison of static versus generative approaches

Many works on declarative bias concentrated on the study of the most efficient bias in terms of the number of hypotheses remaining in the hypothesis space. This criterion can be balanced by another criterion: the *usefulness* of the bias for the end-user of the ILP system. It is clear that there is no general answer, as this criterion depends on the type of application the system is designed for. Thus, the answer to the question "what are the most useful bias for my application", requires that the quality of the bias is not only evaluated on the basis of the number of hypotheses it prunes. Although the "usefulness" including the "comprehensiblity" cannot be yet easily measured without extensive experiments with users, we will try here to give some answer with respect to the representation we have defined within the ILP project.

The distinction between *static* and *generative* approaches seems to be relevant to evaluate the representation on the base of the nature of the knowledge the user has to provide the system with, and the facility for the user to do it. With *static approaches*: a candidate clause belongs to the hypothesis space for learning a program P, if it matches a given schema. Language bias such as input-output modes, range-restriction, etc, may be attached to the schemas (see section 3.1). Clause Set and scheme-based languages typically falls under this category. By contrast, *generative bias* languages do not provide a template to be matched against candidate clauses but, rather, a *set of operators* that incrementally construct clauses. Refinement operators as in NINA [ADRB95] and CLAUDIEN (section 4.2), and MILES-CTL and $\mathcal{HAIKU}$ operators are forms of generative bias. Some forms of analytic learning and Antecedent Description Grammars [Coh93] that use domain theories, also fit into this framework.

- Static approaches requires from the user the description of the set of the possible clauses. This type of requirements seems to be more appropriate for programmers developing Prolog programs and more generally in software development contexts where the developer may have in mind some approximation of the target program. When developing Prolog programs, the programmer will soon reach some syntactic approximation of the desired program where the control structure is more or less determined but the precise input-output behavior may need further work. Programs with mutations, sets of possible clauses and literals, and templates of different kinds seem appropriate for describing such syntactic approximations. However, in this approach, the role of the background knowledge is usually restricted to the vocabulary of the concept description language. As no generality relation on the hypotheses generated from the templates can be easily derived from the language bias they represent, except from the number of literals, most of the static approaches exhaustively explore the hypothesis space without exploiting an order relation to prune it. The gain in comprehensibility of the language bias may thus lead to loose efficiency when searching.

- Generative approaches are appropriate for describing in a concise way and for efficiently exploring very large spaces of possible clauses. In a top-down or bottom-up strategy, for instance, they only require the user's description of an initial example and possibly some vocabulary, the user does not need to provide any hints about the target concept. As a consequence, the control on the form of the hypotheses considered by the system is not as easy as with static approaches. The reason is that there is a large gap separating a generative language and the Prolog clauses that are produced. It may take time for a programmer to determine whether a clause can be generated or not, and a small modification in the bias specification may cause important changes in the space $\mathcal{P}$ of possible programs. In case the solution reached is not satisfying, he has to modify the initial hypothesis, the background knowledge and the search operator without clearly evaluating the effects on the results. This has been called the *unfolding problem* for generative bias languages: the specification of which clauses are possible has to be "unfolded" several times before these clauses are generated. Therefore, the hierarchical structure of the bias specification and the one of the programs that may be learned is different. This approach is thus much less demanding for the user but the form of the results is also more difficult to control. Moreover generative approaches are suitable for exploiting domain theory to guide the s earch, prune the hypothesis space and restrict the vocabulary. For that reason, generative approaches seem to be more appropriate for "classical" Knowledge-Based System development where structured background knowledge may be available and the form of the results is not a priori known than for software development.

4 Interaction of Biases

Although it took time to understand why bias was important for accuracy, its effects on the efficiency of learning is immediate and rather obvious. If there are more possible programs to consider, it may take longer to find one that is acceptable, based on the available examples. But, to this purpose, it is necessary also to consider the search bias and the stopping criterion. The languages of bias described above have been implemented and integrated with search bias and stopping criterion that exploit the capability of the language they are associated with. In particular, Clause Set and $\mathcal{D}$LAB representation are a priori independent of the exploration of the hypothesis space, as opposed to grammars such as GRENDEL's or configuration framework such as $\mathcal{HAIKU}$'s, but, in fact, the search bias they can be associated with is more or less efficient in terms of computational costs depending on the order it will fix for considering the clauses generated from the language.

4.1 Clause Set and Search bias for programs learning

The Clause Sets language, suitable for *program learning* as opposed to single concept learning, has been associated to the system TRACY that is able to learn consistent and correct programs from a set of classified examples. Searching a space of logic programs is more difficult and slower than tuning linear discriminants or growing decision trees, as clauses depend on each other through recursion and contain variables that may be chosen or instantiated in many different ways. This problem of searching a hypothesis space of Logic Programs has been studied and experimented in connection with the learning system TRACY. TRACY is given a set of clauses, and looks for a subset of these clauses that is consistent with the examples, with a complexity that is polynomial with respect to the number of given clauses. TRACY uses a special form of search bias, due to its special way of finding a complete and consistent logic program. The clause set language allows the user to define a set of possible clauses. The relevant aspect here is that this set of clauses comes with an order, and this order is also determined by the user. Intuitively, TRACY's search bias consists in preferring the first clauses in the set, with respect to the given ordering. From the point of view of the user, it is always desirable to describe a language bias with an order where clauses that have a higher probability of being in the target program come first. Therefore, if is often a good thing to split large clause sets and large literal sets, so that user preferences for possible clauses can be made explicit. In fact, expressing such preferences with large literal sets may be ackward, and longer, but simpler, lists of plane clauses may be more appropriate.

4.2 $\mathcal{D}$LAB and Search bias

Together with the clause model formalism $\mathcal{D}$LAB an optimal refinement operator for the system CLAUDIEN has been developed (see [Van93]) that exploit the clause models. This operator non redundantly generates all clauses in the hypothesis space in a general-to-specific way, i.e., it starts from a most general clause allowed by the clause model(s), and systematically searches all nodes (clauses) in the refinement graph, without visiting a node (clause) more than once. A top-down search with θ-subsumption as generality relation is particularly well adapted to $\mathcal{D}$LAB that allows to easily control the number of literals in the generated clauses.

4.3 Language bias properties and search

The approach developed in $\mathcal{HAIKU}$ consists in providing the user with the possibility of both setting static language bias and generative search bias.

The main question arisen by such an approach concerns the best way to combine these two types of bias so that the complexity of the learning process is minimum. Depending on whether it is tested prior to or while searching, a same language bias may require more or less operations.

One scenario for combining static and generative approaches may consist in generating all the allowed clauses from the clause set, and then searching the hypothesis space from an initial hypothesis by applying an operator and testing the candidate hypothesis against *the set of allowed clauses*. Another scenario consists in searching the hypothesis space in a generative way and check if the generated hypotheses verify the language bias before testing them against the examples. None of these scenarii solves the following problem: if static language bias and generative search bias are set independently, their effects may be redundant and then the system may have to test unuseful restrictions. For instance, range-restriction as language bias is redundant with inversion of resolution as search bias with range-restricted initial clause and domain theory.

Finally, an efficient combined approach implemented in $\mathcal{HAIKU}$ system consists in first excluding redundant language bias and then applying some language bias prior to searching and the others while searching, so that their application may be as less costly as possible. This require to precisely identify the interaction between biases, independently of their representation.

4.3.1 Language bias properties

The analysis of the interaction between language bias and search operators leads to define the following properties of the language bias with respect to the hypothesis space generated by the search operator (see [LRI95] for more details).

Notation 4.1 *Given a language bias B, the set of clauses satisfying B is denoted $\mathcal{L}_B$.*

Definition 1 (Private bias) *A bias B is private with respect to an operator $\mathcal{O}$ iff for all hypotheses H and H' so that $H' \in \mathcal{O}^*(H)$, $H \notin \mathcal{L}_B \Rightarrow H' \notin \mathcal{L}_B$.*

For instance, the limitation on the size of the body of the concept definition is a private bias for the adding-literal operator. Symmetrically,

Definition 2 (Closed bias) *A bias B is closed with respect to an operator $\mathcal{O}$ iff for all hypotheses H and H' so that $H' \in \mathcal{O}^*(H)$, $H \in \mathcal{L}_B \Rightarrow H' \in \mathcal{L}_B$.*

For instance, range-restriction is a closed bias for the absorption operator with a range-restricted theory.

4.3.2 Exploitation of the properties

Depending on the properties of a given language bias B (e.g. private or closed), some redundant tests of bias and generation of hypotheses can be avoided.

- B is *both closed and private* w.r.t. the operator $\mathcal{O}$. If a given hypothesis H satisfies B, any hypothesis generated from H will also satisfy B. It is therefore unnecessary to test B on further alterations of H by $\mathcal{O}$. If a given hypothesis H does not satisfies B, any hypothesis generated from H will not satisfy B, it is thus unuseful to generate new hypotheses from H.

- B is *only closed* w.r.t. the operator $\mathcal{O}$. If a given hypothesis H satisfies B, any hypothesis generated from H will also satisfy B.It is therefore unnecessary to test B on further alteration of H by $\mathcal{O}$. If not, the hypotheses generated from H have to be tested against B.

- B is *only private* w.r.t. the operator $\mathcal{O}$. If a given hypothesis H does not satisfy B it is thus unuseful to generate new hypotheses from H. If it does, the hypotheses generated from H have to be tested against B.

- B is *neither closed nor private*w.r.t. the operator $\mathcal{O}$. The hypotheses generated from H have to be tested against B.

These conclusions lead to the following learning scenario. The initial bound or, starting clauses have to be tested against biases that are both closed and private. Then each generated hypothesis has to tested against private biases. Finally, when a clause is learned as part of the target program, it has to be tested against not private biases before being validated. In any case, a hypothesis that failed when being tested against language biases is left.

4.3.3 Properties of classical language biases

Table 2 shows that there are few closed or private biases and no private *and* closed bias for the dropping-literal operator (the same observation for the adding-literal operator may be found in [LRI95]). The absorption operator is a combination of the adding-

Language Bias	**Closed Bias**	**Private Bias**
Number of literals	X	
Number of variables	X	
Depth	X	
Degree		
Range-restriction		X
Connexion		
Functionality UI		X
Functionality IO		X
Functionality UO		
Functionality OO	X	
Functionality II		
Total	4	3

Table 2: Bias properties for dropping-literal

Language Bias	**Closed Bias**	**Private Bias**
Number of literals	X	
Number of variables	*no*	
Depth		
Degree	*no*	
Range-restriction		X
Connection		
Functionnality UI		X
Functionnality IO		X
Functionnality UO		
Functionnality OO		
Functionnality II		
Total	1	3

Table 3: Bias properties for absorption

and dropping-literal operators. Thus, one could predict that the absorption operator will not exhibit any private or closed properties. However some useful properties remain as shown in table 3. For instance, the biases *Number of existential variables* and *Degree* are closed only if the domain theory is range-restricted. This can be explained by the relationship defined in BK, between literals that are dropped and added by the absorption operator.

5 Conclusion

The results that have been achieved within the ILP project are considerable with respect to the initial state of the art. Not only general and reusable typologies of bias have been defined along the line by previous works but also models of learning have been developed that elicit the role of bias with respect to the input knowledge and the learning operations. Based on this conceptual representation, operationalisation has been achieved through two kinds of representation of bias in learning systems: language of bias that are interpreted by systems in the form of tests, and bias configuration, that can be viewed as a a pre-compilation of bias into pieces of software that are assemblied for bias setting. General properties of bias have been identified for both type of methods, that first provide guidelines for choosing a representation according to the learning task to achieve and second, provide clear rules for an efficient bias setting. As a next step, extensive experiments in various domains with the implemented systems and theoretical study on the effects of combinations of biases should allow to relate in a more general way application requirements and bias setting and shifting.

Acknowledgements

This work is supported by ILP BRA 6020.

References

[ADRB95] H. Adé, L. De Raedt, and M. Bruynooghe. Declarative Bias for Specific-To-General ILP Systems. *Machine Learning*, 1995.

[Ber93] F. Bergadano. Towards an inductive logic programming language. Technical Report ESPRIT project no. 6020 ILP Deliverable TO1, Computer Science Department, University of Torino, 1993.

[BG95] F. Bergadano, D. Gunetti. *Inductive Logic Programming: from Machine Learning to Software Engineering* MIT Press, 1995.

[BJ93] P. Brazdil and A. Jorge. Exploiting algorithm sketches in ILP. In *Proc. of 3rd International Workshop on Inductive Logic Programming ILP-93*, Technical Report, IJS-DP-6707. J. Stefan Institute, 1993.

[Coh93] W.W. Cohen. Rapid prototyping of ILP systems using explicit bias. In *Proc. of IJCAI-93 Workshop on Inductive Logic Programming.* Morgan Kaufmann, 1993.

[Cov69] T. M. Cover. Learning in Pattern Recognition. In S. Watanabe, editor, *Methodologies of Pattern Recognition*, New York, 1969. Academic Press.

[DB92] L. De Raedt and M. Bruynooghe. A unifying framework for concept-learning algorithms. *The Knowledge Engineering Review*, 7(3):251–269, 1992.

[DR95] L. Dehaspe and L. De Raedt, A declarative language bias for concept learning and knowledge discovery engines. *Report CW 214, Dept. of Computer Science*, Katholieke Universiteit Leuven, 1995.

[Dev88] L. Devroye. Automatic Pattern Recognition: A Study of the Probability of Error. *IEEE Trans. on PAMI*, 10(4):530–543, 1988.

[DR92] L. De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach.* Academic Press, 1992.

[EHR83] W. Emde, C.U. Habel, and C.R. Rollinger. The discovery of the equator or concept driven learning. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 455–458. Morgan Kaufmann, 1983.

[KW92] J.-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming.* Academic Press, 1992.

[KMS92] R. King, S. Muggleton, and M. Sternberg. Protein secondary structure prediction using logic. In *Proc. 2nd Int. Workshop on ILP*, Tokyo, Japan, 1992.

[Kli94] V. Klingspor. GRDT: Enhancing model-based learning for its application in robot navigation. LS-8 report 5, Universität Dortmund, 1994.

[LB85] H.J. Levesque, R.J. Brachman A Fundamental Tradeoff in Knowledge Representation and Reasoning. In *Readings in Knowledge Representation*, R.J. Brachman and H.J. Levesque (Ed.), Morgan Kaufmann, 1985.

[LRI95] C. Nedellec, C. Rouveirol, F. Torre First results on interactions of language and search bias in ILP Deliverable LRI2.c, ILP Project, 1995.

[Mic83b] R. S. Michalski. A theory and methodology of inductive learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach.* Tioga, 1983.

[Mit80] T. M. Mitchell. The need for biases in learning generalizations. Technical Report TR-117, Department of Computer Science, Rutgers University, 1980.

[SMK94] A. Srinavasasan, S. Muggleton, R. King and M. Sternberg. Mutagenesis: IlP experiments in a non-determinate biological domain. *proceedings of the Fourth International Workshop on ILP* S. Wrobel (ed.), 1994.

[MWKE94] K. Morik, S. Wrobel, J. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning: Theory Methods and Applications.* Academic Press, 1994.

[NR94] C. Nedellec and C. Rouveirol *Specification of the HAIKU system.* Technical report 928, University Paris-Sud, 1994.

[RG90] S. Russell and B. Grosof. Declarative Bias: An Overview In P.D.Benjamin, editor, *Change of representation and inductive bias.* Kluwer Academic Publishers, 1990.

[RG90] S. Russell and B. Grosof. A sketch of autonomous learning using declarative bias. In P.B. Brazdil and K. Konolige, editors, *Machine Learning, Meta-Reasoning and Logics*, pages 19–54. Kluwer Academic Publishers, 1990.

[Rou94] C. Rouveirol and P. Albert Knowledge Level Model of a configurable Learning System. in *Proc. of the eigth European Knowledge Acquisition Workshop*, LNAI 867, Steels, L. and Schreiber, G. and Van de Velde, W. (Ed.), Springer Verlag, 1994.

[SB93] L. Saitta and F. Bergadano. Pattern Recognition and Valiant's Learning Framework. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(2), 1993.

[SP91] G. Silverstein and M. Pazzani. Relational clichés: Constraining constructive induction during relational learning. In *Proc. of the Eighth International Conference on Machine Learning, Evanston.* Morgan Kaufmann, 1991.

[Tau92] B. Tausend. Using and adapting schemes for the induction of Horn clauses. In *ECAI Workshop Logical Approaches to Machine Learning*, Wien, 1992.

[Tau94a] B. Tausend. Representing biases for inductive logic programming. In *ECML-94, European Conference on Machine Learning*, Catania, Italy. Springer, 1994.

[Tau94b] B. Tausend. *Beschränkungen der Hypothesensprache und ihre Repräsentation in der ILP.* PhD thesis, Universität Stuttgart, 1994.

[Tau95a] B. Tausend. A guided tour through hypothesis spaces in ILP. In *ECML-95, European Conference on Machine Learning*, Heraklion, Greece. Springer, 1995.

[Tor95] F. Torre *Exploitation de biais de language en apprentissage symbolique automatique dans un cadre logique.* Rapport de DEA, Université Paris-Sud, 1995.

[Utg86] P.E. Utgoff. Shift of bias for inductive concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine learning: An artificial intelligence approach.* Morgan Kaufmann, 1986.

[Van93] W. Vanlaer. Inductief Afleiden van Logische Regels. Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1993. in Dutch.

[VC81] V. N. Vapnik and Y. A. Chervonenkis. Necessary and Sufficient Conditions for the Uniform Convergence of Means to Their Expectations. *Theory of Probability and Its Applications*, 26:532–553, 1981.

[VLDDR94] W. Van Laer, L. Dehaspe, and L. De Raedt. Applications of a logical discovery engine. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 263–274, 1994.

[WO91] R. Wirth and P. O'Rorke. Constraints on predicate invention. In *Eighth International Conference on Machine Learning.* Morgan Kaufmann, 1991.