Inductive Logic Programming: derivations, successes and shortcomings

Stephen Muggleton
Oxford University Computing Laboratory,
11 Keble Road,
Oxford,
OX1 3QD,
United Kingdom.

Abstract

Inductive Logic Programming (ILP) is a research area which investigates the construction of quantified definite clause theories from examples and background knowledge. ILP systems have been applied successfully in a number of real-world domains. These include the learning of structure-activity rules for drug design, finite-element mesh design rules, rules for primary-secondary prediction of protein structure and fault diagnosis rules for satellites. There is a well established tradition of learning-in-the-limit results in ILP. Recently some results within Valiant's PAC-learning framework have also been demonstrated for ILP systems. In this paper it is argued that algorithms can be directly *derived* from the formal specifications of ILP. This provides a common basis for Inverse Resolution, Explanation-Based Learning, Abduction and Relative Least General Generalisation. A new general-purpose, efficient approach to predicate invention is demonstrated. ILP is underconstrained by its logical specification. Therefore a brief overview of extra-logical constraints used in ILP systems is given. Some present limitations and research directions for the field are identified.

1 Introduction

The framework for Inductive Logic Programming (ILP) [37, 38] is one of the most general within the field of Machine Learning. ILP systems construct concept definitions (logic programs) from examples and a logical domain theory (background knowledge). This goes beyond the more established *empirical learning* framework [32, 48, 5, 6] because of the use of a quantified relational logic together with background knowledge. It goes beyond the *explanation-based learning* framework [33, 11] due to the lack of insistence on complete and correct background knowledge.

The use of a relational logic formalism has allowed successful application of ILP systems in a number of domains in which the concepts to be learned cannot easily be described in an attribute-value language. These applications include structure-activity prediction for drug design [25, 57], protein secondary-structure prediction [42], and finite element mesh design [12]. It is worth comparing these results with existing *scientific discovery* systems in machine learning. By normal scientific standards it does not make sense to call BACON's [27] and AM's [29] achievements scientific/mathematical discovery since they did not produce new knowledge refereed and published in the journals of their subject area. The above applications of drug design and protein folding *did* produce machine-derived new knowledge, published in top scientific journals. There are very few other examples within AI where this has been achieved.

The generality of the ILP approach has allowed many exciting new types of application domain. In addition to the above real-world application areas ILP systems such as MIS [55], Marvin [54], CIGOL [44], ML-SMART [3], FOIL [50], Golem [41], ITOU [52], RDT [24], CLINT [9], FOCL [46], SIERES [61] and LINUS [14] are all capable of synthesising logic programs containing recursion. They can also deal with domains containing explicit representation of time [16] and learn grammar rules for natural language processing [60].

Learning-in-the-limit results are well-established in the ILP literature both for full-clausal logic [47] and definite clause logic [1, 9]. However, these results tell one little about the efficiency of learning. In contrast, Valiant's [59] PAC (Probably-Approximately-Correct) framework is aimed at providing complexity results for machine learning algorithms. However, Haussler's [21] negative PAC result concerning existentially quantified formulae seemed initially to exclude the possibility of PAC results for quantified logic. The situation has been improved by recent positive results in significant sized subsets of definite clause logic. Namely, single constrained Horn clauses [45] and k-clause ij-determinate logic programs [15]. Recent results by Kietz [23] indicate that every proper superset of the k-clause ij-determinate language is not PAC learnable. This seems to indicate a ceiling to extensions of present approaches.

As the ILP applications areas show, Horn clause logic is a powerful representation language for concept learning. It also has a clear model-theoretic semantics which is inherited from Logic Programming [30]. However, with the generality of the approach come problems with searching a large hypothesis space. A clear logical framework helps in deriving efficient algorithms for constraining and searching this space. In Section 2 the formal definitions of ILP are used to derive existing specific-general and general-specific algorithms. Additionally, in Section 2.5 a new method for carrying out predicate invention is derived in this way. In Section 3 extralogical constraints used within existing ILP systems are discussed. In Section 4 some of the shortcomings of existing ILP systems are discussed and potential remedies suggested.

2 Formal logical setting for ILP

One might ask why ILP should need a very formal definition of its logical setting? After all, Machine Learning research has progressed quite happily without much formal apparatus. Surely formalisation is time-consuming and impedes progress in implementing systems? This paper argues the opposite. Without formalisation it is not clear what one is trying to achieve in an implementation. Techniques from one implementation cannot be transferred easily to another. However, this section will demonstrate a more direct and immediate advantage. That is, if a small number of formulae are used to define the high level properties of a learning system it is often possible to manipulate these formulae algebraically to derive a complete and correct algorithm which satisfies them.

2.1 The setting

The usual context for ILP is as follows. The learning agent is provided with background knowledge B, positive examples E^+ and negative examples E^- and constructs an hypothesis H. B, E^+ E^- and H are each logic programs. A logic program is a set of definite clauses each having the form

$$h \leftarrow b_1, b_2, \ldots$$

where h is an atom and $b_1, b_2, \ldots$ is a set of atoms. Usually E^+ and E^- contain only ground clauses, with empty bodies. The following symbols are used below: $\wedge$ (logical and), $\vee$ (logical or), $\vdash$ (logically proves), $\square$ (Falsity). The conditions for construction of H are

Necessity: $B \nvdash E^+$

Sufficiency: $B \wedge H \vdash E^+$

Weak consistency: $B \wedge H \nvdash \square$

Strong consistency: $B \wedge H \wedge E^- \nvdash \square$

Note that *strong consistency* is not required for systems that deal with noise (eg. Golem, FOIL and LINUS). The four conditions above capture *all* the logical requirements of an ILP system. Both *Necessity* and *Consistency* can be checked using a theorem prover. Given that all formulae involved are Horn clauses, the theorem prover used need be nothing more than a Prolog interpreter, with some minor alterations, such as iterative deepening, to ensure logical completeness.

2.2 Deriving algorithms from the specification of ILP

The *sufficiency* condition captures the notion of generalising examples relative to background knowledge. A theorem prover cannot be directly applied to derive H from B and E^+. However, by simple application of the Deduction Theorem the *sufficiency* condition can be rewritten as follows.

Sufficiency*: $B \wedge \overline{E^+} \vdash \overline{H}$

This simple alteration has a very profound effect. The negation of the hypothesis can now be deductively derived from the negation of the examples together with the background knowledge. This is true no matter what form the examples take and what form the hypothesis takes. So, in order to understand the implications of *sufficiency** better, in the following sections it is shown how different algorithms, for different purposes can be derived from this relation.

2.3 Single example clause, single hypothesis clause

This problem has been studied extensively by researchers investigating both EBL [33, 11], Inverse Resolution [54, 1, 44, 37, 52, 9] and Abduction [26, 31]. For simplicity, let us assume that both the example and the hypothesised clause are definite clauses. Thus

$$\begin{aligned} E^+ &= h \leftarrow b_1, b_2, \ldots = h \vee \overline{b_1} \vee \overline{b_2} \ldots \\ H &= h' \leftarrow b_1', b_2', \ldots = h' \vee \overline{b_1'} \vee \overline{b_2'} \ldots \end{aligned}$$

Now substituting these into *sufficiency** gives

$$\begin{aligned} B \wedge \overline{(h \vee \overline{b_1} \vee \overline{b_2} \ldots)} &\vdash \overline{(h' \vee \overline{b_1'} \vee \overline{b_2'} \ldots)} \\ B \wedge \overline{h} \wedge b_1 \wedge b_2 \ldots &\vdash \overline{h'} \wedge b_1' \wedge b_2' \ldots \end{aligned}$$

Note that $\overline{h}, \overline{h'}, b_i$ and b_i' are all ground skolemised literals. Suppose we use $B \wedge \overline{E^+}$ to generate *all* ground unit clause consequences. When negated, the resulting (possibly infinite[1]) clause is a unique, most specific solution for all hypotheses which fit the *sufficiency* condition. All such clauses entail this clause. Thus the entire hypothesis space can be generated by dropping literals, variabilising terms or inverting implication [28, 39, 22]. No matter what control method is used for searching this space (general-specific or specific-general), all algorithms within EBL and Inverse Resolution are based on the above relationship. This is shown in detail for Inverse Resolution in [37].

What happens when more than one negative literal is a ground consequence of $B \wedge \overline{E^+}$? In the general case, the most specific clause will then be $h_1' \vee h_2', .. \vee \overline{b_1'}, \vee \overline{b_2'} \ldots$. If the hypothesis is required to be a definite clause, the set of most-specific solutions is

$$\begin{aligned} h_1' &\leftarrow b_1', b_2', \ldots \\ h_2' &\leftarrow b_1', b_2', \ldots \\ &\ldots \end{aligned}$$

h_1' and h_2' may not have the same predicate symbol as h in the example. This set of most specific clauses, representing a set of hypothesis spaces, can be seen

[1] Specific-general ILP algorithms such as CLINT [9] Golem [41] use contrained subsets of definite clause logic to ensure finiteness of the most-specific clause.

as the basis for abduction, theory revision [51] and multiple predicate learning [10].

Example 1 *Let*

$$B = \begin{cases} haswings(X) \leftarrow bird(X) \\ bird(X) \leftarrow vulture(X) \end{cases}$$

$$E^+ = haswings(tweety) \leftarrow$$

The ground unit consequences of $B \wedge \overline{E^+}$ *are*

$$C = \overline{bird(tweety)} \wedge \overline{vulture(tweety)} \wedge \overline{haswings(tweety)}$$

This leads to 3 most-specific starting clauses.

$$H = \begin{cases} bird(tweety) \leftarrow \\ vulture(tweety) \leftarrow \\ haswings(tweety) \leftarrow \end{cases}$$

If any one of these clauses is added to B *then* E^+ *becomes a consequence of the new theory.*

2.4 Multiple examples, single hypothesis clause

This problem is faced in general-specific learning algorithms such as MIS [55], FOIL [50], RDT [24] as well as specific-general algorithms such as Golem [41]. Let us assume that $E^+ = e_1 \wedge e_2 \wedge \ldots$ is a set of ground atoms. Suppose C denotes the set of unit consequences of $B \wedge \overline{E^+}$. From *sufficiency** it is clear that

$$B \wedge \overline{E^+} \vdash \overline{E^+} \wedge C$$

Substituting for E^+ and rearranging gives

$$B \wedge \overline{E^+} \vdash (\overline{e_1} \vee \overline{e_2} \vee \ldots) \wedge C$$
$$B \wedge \overline{E^+} \vdash (\overline{e_1} \wedge C) \vee (\overline{e_2} \wedge C) \vee \ldots$$

Therefore $H = (e_1 \vee \overline{C}) \wedge (e_2 \vee \overline{C}) \wedge \ldots$, which is a set of clauses. Since the solution must be a single clause, systems such as Golem construct the most specific clause which subsumes all these clauses. General-specific algorithms search the set of clauses which subsume subsets of these clauses, starting from the empty clause. If the hypothesis is a set of two or more clauses, then again each clause in this set subsumes the set of most-specific clauses above. FOIL assumes explicit pre-construction of the ground atoms in C to speed subsumption testing.

Example 2 *Let*

$$B = \begin{cases} father(harry, john) \leftarrow \\ father(john, fred) \leftarrow \\ uncle(harry, jill) \leftarrow \end{cases}$$

$$E^+ = \begin{cases} parent(harry, john) \leftarrow \\ parent(john, fred) \leftarrow \end{cases}$$

The ground unit consequences of $B \wedge \overline{E^+}$ *are*

$$C = father(harry, john) \wedge father(john, fred) \wedge uncle(harry, jill)$$

This leads to the following most specific clauses

$$e_1 \vee \overline{C} = parent(harry, john) \leftarrow father(harry, john), father(john, fred), uncle(harry, jill)$$
$$e_2 \vee \overline{C} = parent(john, fred) \leftarrow father(harry, john), father(john, fred), uncle(harry, jill)$$

The least general generalisation is then

$$lgg(e_1 \vee \overline{C}, e_2 \vee \overline{C}) = parent(A, B) \leftarrow father(A, B), father(C, D)$$

2.5 Single example, multiple clause hypothesis (predicate invention)

The *sufficiency** condition can be used for any form of hypothesis construction. Thus it should be possible to derive how *predicate invention* (introduction of new predicates) is carried out with this relationship. Let us first define predicate invention more formally. If P is a logic program then the set of all predicate symbols found in the heads of clauses of P is called the definitional vocabulary of P or $V(P)$. ILP has the following three definitional vocabularies.

Observational vocabulary: $\mathcal{O} = V(E^+ \cup E^-)$

Theoretical vocabulary: $\mathcal{T} = V(B) - \mathcal{O}$

Invented vocabulary: $\mathcal{I} = V(H) - (\mathcal{T} \cup \mathcal{O})$

An ILP system is said to carry out *predicate invention* whenever $I \neq \emptyset$.

Most specific predicate invention can be carried out using the rule of And-introduction (conversely Or-introduction [22]). These logical equivalences are as follows.

And-introduction: $X \equiv (X \wedge Y) \vee (X \wedge \overline{Y})$

Or-introduction: $X \equiv (X \vee Y) \wedge (X \vee \overline{Y})$

Note that the predicate symbols in Y can be chosen arbitrarily and may be quite distinct from those in X. Now letting C be the set of all unit consequences of $B \wedge \overline{E^+}$ and using And-introduction gives

$$B \wedge \overline{E^+} \vdash C$$
$$B \wedge \overline{E^+} \vdash (p \wedge C) \vee (\overline{p} \wedge C)$$

where p is a ground atom whose predicate symbol is not in $(\mathcal{T} \cup \mathcal{O})$. Thus H is any set of clauses which entails $(\overline{p} \vee \overline{C}) \wedge (p \vee \overline{C})$. This can be viewed as the introduction of a clause head (p) and a calling atom from the body of a clause $(\overline{p})$. All methods of predicate invention, such as those using the W-operator [44], construct clauses which entail the above forms of clauses. However, there is an infinite set of atoms p which could be And-introduced in this way. Each of these represents the invention of a different predicate. In [40] it is shown how these invented predicate can be arranged in a partially-ordered lattice of utility with a unique top $(\top)$ and bottom $(\bot)$ element. Within this lattice invented predicates are unique up to renaming of the predicate symbol and re-ordering of arguments. This provides for a unique p, which is an instance of $\bot$ to be introduced which simply contains the set of all ground terms in C. Clauses can then be generalised through the relative clause refinement lattice by dropping arguments from p or generalising $C \vee p$ and $C \vee \overline{p}$.

Example 3 *The following example involves inventing 'lessthan' in learning to find the minimum element of a list. Let*

$$B = min(X, [X]) \leftarrow$$
$$E^+ = min(2, [3, 2]) \leftarrow$$

The ground unit consequences of $B \wedge \overline{E^+}$ *are*

$$\overline{min(2, [3, 2])} \wedge min(2, [2]) \wedge min(3, [3])$$

Let $p = p1(2, 3, [2], [3], [3, 2])$. *This leads to the following 2 most-specific starting clauses for predicate invention.*

$$H = \begin{cases} min(2, [3, 2]) \leftarrow min(2, [2]), min(3, [3]), p1(2, 3, [2], [3], [3, 2]) \\ p1(2, 3, [2], [3], [3, 2]); min(2, [3, 2]) \leftarrow min(2, [2]), min(3, [3]) \end{cases}$$

Generalising and renaming the predicate symbol gives

$$H' = \begin{cases} min(Y, [Y|Z]) \leftarrow min(X, Z), lessthan(Y, X) \\ lessthan(2, 3) \leftarrow \end{cases}$$

3 Extralogical constraints in ILP

In the previous section only the logical constraints used in ILP systems were discussed. It was shown that these can be usefully manipulated to derive the skeleton of an ILP system. However, in order to ensure efficiency, it is usually found necessary to employ extra-logical constraints within ILP systems. This is done in two complementary ways: statistical confirmation and language restrictions (bias). Confirmation theory fits a graded preference surface to the hypothesis space, while language restrictions reduce the size of the hypothesis space. In the following two subsections ILP developments in these areas are discussed.

3.1 Statistical confirmation

Philosophy of Science uses the notion of a confirmation function [19] to give a grading of preferred hypotheses. A confirmation function is a total function that maps elements of the hypothesis space onto a subset of the real numbers. Within ILP confirmation functions based on concepts from Algorithmic Complexity Theory and Minimal Description Length Theory have been developed [36, 50, 43, 8]. Confirmation functions based on Bayesian statistical analysis have also been found useful in handling noise in ILP real-world domains [13].

In [56] the authors explore the use of upper and lower bound estimates of a confirmation function to guide multi-layered predicate invention. The resulting algorithm is a non-backtracking version of an $A*$ search. This approach is effective for guiding "deep" predicate invention, with multiple layers.

3.2 Language restrictions (bias)

Recent results in PAC-learning [45, 15, 23] show that reducing the size of the target language often makes ILP learning more tractable. The main restrictions are on the introduction of existentially quantified variables in the bodies of definite clauses. CLINT [9] places a finite limit on the number of such variables that are allowed to be introduced. Golem [41] requires that the quantification of such variables is limited to Hilbert ϵ^* quantification (exists at most one) and that these "determinate" variables be introduced into the clause body in a fixed number of at most i layers. FOIL [49] has since also made use of the determinate restriction introduced first in Golem.

An alternative approach to language restriction involves the provision of declarative templates which describe the form hypotheses must take. For instance, the algorithm may be told the hypothesis takes the form

$$Q(S1) \leftarrow P1(S1), preceding\_state(S1, S0), P2(S0)$$

where $Q, P1, P2$ can be instantiated with any predicate symbols from the background knowledge. This approach is sometimes referred to as "rule-models" [24], but can also be viewed as learning by analogies expressed as higher-order logic

Restriction	Systems	Problematic Domains
Ground background knowledge	FOIL, Golem	Qualitative, chess, natural language
Non-numerical data	ITOU, FOIL, Golem, SIERES, CLINT, RDT	Meshes, drugs
Determinacy	Golem, FOIL, LINUS	Qualitative, chess meshes
Search myopia	FOIL, FOCL	List&number theory,
Efficiency of learning	ITOU, CLINT	Proteins, chess, satellites

Figure 1: Restrictions that have led to problems in real-world applications

defaults [18, 20, 9]. This has led to some interest within ILP in being able to learn higher-order predicates [17].

Related to the idea of rule-models is the use of mode and type declarations in MIS, Golem, SIERES, FOIL and LINUS. A general scheme of using mode declarations is under development by the author in the ILP system Progol. The mode declarations for Progol take the following form.

mode(1,append(+list,+list,-list))
mode(,append(-list,-list,+list))*

The first mode states that append will succeed once (1) when the first two arguments are instantiated with lists and on return the third argument will be instantiated by a list. Types such as 'list' are user-defined as monadic background predicates. The second declaration states that append will succeed finitely many times (*) with the third argument instantiated as a list. The specified limit on the degree of indeterminacy of the call can be any counting number or '*'.

In [7, 2] the concept of "rule-models" is further generalised to that of an hypothesis space language specified by a set of grammar rules. This approach provides a general purpose "declarative bias" and is reminiscent of earlier work on "determinations" [53]. Although determinations in their present form are restricted to propositional logic learning, they have been proved to have a dramatic effect on reducing learning complexity [53].

4 Shortcomings of ILP systems

Despite the rapid development of the ILP research area there is some way to go before ILP could deliver a technology that would be used widely by working scientists and engineers. Figure 1 lists restrictions that certain ILP systems have that have led to awkwardness in applying them to real-world applications. The domains referred to cryptically in the table are, in alphabetical order, the following

Chess. Learning endgame strategies [35].

Drugs. Structure-activity prediction [25].

List&number theory. Quick-sort, multiply, etc. [41].

Meshes. Rules for Finite Element Mesh design [12].

Natural language. Grammar acquisition [60].

Proteins. Secondary-structure prediction [42].

Qualitative. Learning qualitative models [4].

Satellites. Temporal fault diagnosis. [16].

In the following subsections some approaches to avoiding these restrictions will be sketched. The problems encountered in applications will be explained and some remedies suggested.

4.1 Ground background knowledge

Golem and FOIL require all background knowledge to be given extensionally in tabular form. This is acceptable and very efficient when the number of ground instances required is small. In domains such as qualitative model construction, chess and natural language this is not feasible. Effective learning algorithms need to be able to call a Prolog interpreter to derive ground atoms from intensionally-coded specifications of background predicates. To do so they should only derive background atoms that are relevant to the examples. CLINT, ITOU and LINUS all achieve these aims to varying degrees.

4.2 Non-numerical data

The mesh domain involves predicting the number of sections that an edge of a CAD object should be broken into for efficient finite-element analysis. The rules developed by Golem thus have the following form.

$$mesh(Obj, 8) \leftarrow connected(Obj, Obj1), \ldots$$

However with a small number of examples it is hard to get enough examples in which the prediction is an exact number, such as 8. Instead we would like the rules to predict an interval such as

$$mesh(Obj, X) \leftarrow 7 \leq X \leq 9, connected(Obj, Obj1), \ldots$$

This kind of construction is not handled elegantly by existing systems. In statistics this problem of numerical prediction is known as regression. Many efficient statistical algorithms exist for handling numerical data. ILP system designers might do well to look at smoothly integrating such approaches into their systems. Recent work on introducing linear inequalities into inductively constructed definite clauses [34] provides an elegant logical framework for this problem.

4.3 Determinacy

The ij-determinate restriction is both powerful and widely used. However, it is very unnatural for many domains. Consider the following chess strategy clause.

$$won(Position, black) \leftarrow move(Position, Position1), \ldots$$

Clearly there will usually be many valid substitutions for Position1. This problem comes up whenever the objects in the domain represent nodes in a connected graph. This is precisely the kind of problem in which ILP algorithms should be more easily applied than attribute-value systems. Kietz's result [23] indicates that there may not be any general PAC solution to learning non-determinate logic programs.

4.4 Search myopia

This problem is an inherent weakness of heuristically-guided general-specific clause construction systems such as FOIL and FOCL. Consider the following recursive clause for multiplication.

$$mult(A, B, C) \leftarrow succ(A, D), mult(D, B, E), plus(E, B, C).$$

The original FOIL [50] could not learn this clause because with a partially developed clause, none of the atoms in the body make a distinction between positive and negative instances. Only the entire set of three atoms together have a non-zero "gain". FOIL2 [49] overcame this problem by introducing all zero-gain determinate literals at the beginning. This gives FOIL2 a mixed general-specific and specific-general control strategy. However, the problem now simply recedes to non-determinate clauses with the same property. For instance, consider the following clause concerning graphs.

$$\begin{aligned} threeloop(Node) \leftarrow \quad & edge(Node, Node1), edge(Node1, Node2), \\ & edge(Node2, Node) \end{aligned}$$

When FOIL2 tries to construct this clause, each 'edge' literal will again have zero gain. Since the atoms are nondeterminate FOIL2 will fail. This form of myopia is not encountered by specific-general algorithms such as CLINT which start with all relevant literals and prune out unnecessary ones.

4.5 Efficiency of learning

One of the most demanding problems for ILP system developers is that of efficiency. Many interesting real-world problems, such as the protein prediction problem, involve thousands or even millions of examples. Scaling ILP systems to deal with such large databases is a non-trivial task. It may be that methods such as "windowing", successfully applied in ID3, could be incorporated into ILP systems.

5 Conclusion and future directions

Inductive Logic Programming is a fast-growing research area. The last few years have seen the area of quantified logic learning develop from a theoretical backwater into a mainstream applied research area. Many of the problems encountered on the way can make use of solutions developed in Machine Learning, Statistics and Logic Programming.

It should be clear from Section 2 that logical theorem-proving is at the heart of all ILP methods. For this reason it must be worth asking whether the technology of Prolog interpreters is sufficient for all purposes. Reconsider Example 1 in Section 2.3. Implementing a general system that carried out the inference in this example would require a full-clausal theorem prover. Is it worth going to this more computationally expensive thechnique? Luc de Raedt has recently started investigating the new generation of efficient full-clausal theorem-provers such as that described by Stickel [58]. Stickel's theorem prover compiles full clauses into a set of definite clauses. These definite clauses are then executed by a Prolog interpreter using iterative deepening. This technique maintains most of Prolog's efficiency while allowing full theorem proving. Full theorem proving is also useful for implementing constraint checking in ILP systems [9]. Learning full-clausal theories is a largely unexplored and exciting new area for ILP.

ILP research has many issues to deal with and many directions to go. By maintaining strong connections between theory, implementations and applications, ILP has the potential to develop into a powerful and widely-used technology.

Acknowledgements.

The author would like to thank Luc de Raedt for helpful and interesting discussions on the topics in this paper. This work was supported by the Esprit Basic Research Action ILP, project 6020.

References

[1] R.B. Banerji. Learning in the limit in a growing language. In *IJCAI-87*, pages 280–282, San Mateo, CA, 1987. Morgan-Kaufmann.

[2] F. Bergadano. Towards an inductive logic programming language. Technical report, University of Torino, Torino, Italy, 1992.

[3] F. Bergadano and A. Giordana. Guiding induction with domain theories. In Y. Kodratoff and R. Michalski, editors, *Machine learning: an artificial intelligence approach*, volume 3, pages 474–492. Morgan Kaufmann, San Mateo, CA, 1990.

[4] I. Bratko, S. Muggleton, and A. Varsek. Learning qualitative models of dynamic systems. In *Proceedings of the Eighth International Machine Learning Workshop*, San Mateo, Ca, 1991. Morgan-Kaufmann.

[5] B. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: a knowledge-elicitation tool for sophisticated users. In *Progress in machine learning*, pages 31–45, Wilmslow, England, 1987. Sigma.

[6] P. Clark and T. Niblett. The CN2 algorithm. *Machine Learning*, 3(4):261–283, 1989.

[7] W. Cohen. Compiling prior knowledge into an explicit bias. In D. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning*, pages 102–110. Morgan Kaufmann, San Mateo: CA, 1992.

[8] D. Conklin and I. Witten. Complexity-based induction. Technical report, Dept. of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, 1992.

[9] L. de Raedt. Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8:107–150, 1992.

[10] L. de Raedt, N. Lavrac, and S. Dzeroski. Multiple predicate learning. CW 65, Dept. of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, 1992.

[11] G. DeJong. Generalisations based on explanations. In *IJCAI-81*, pages 67–69, San Mateo, CA, 1981. Morgan-Kaufmann.

[12] B. Dolsak and S. Muggleton. The application of Inductive Logic Programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*, London, 1992. Academic Press.

[13] S. Dzeroski. *Handling noise in Inductive Logic Programming.* PhD thesis, :University of Ljubljana, 1991.

[14] S. Dzeroski and N. Lavrac. Refinement graphs for FOIL and LINUS. In S. Muggleton, editor, *Inductive Logic Programming.* Academic Press, London, 1992.

[15] S. Dzeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *COLT 92: Proceedings of the Conference on Learning Theory*, San Mateo, CA, 1992. Morgan-Kaufmann.

[16] C. Feng. Inducing temporal fault dignostic rules from a qualitative model. In S. Muggleton, editor, *Inductive Logic Programming.* Academic Press, London, 1992.

[17] C. Feng and S. Muggleton. Towards inductive generalisation in higher order logic. In D. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning*, pages 154–162. Morgan Kaufmann, San Mateo: CA, 1992.

[18] D. Gentner. Structure-mapping: a theoretical framework for analogy. *Cognitive Science*, 7:155–170, 1983.

[19] D. Gillies. Confirmation theory and machine learning. In *Proceedings of the Second Inductive Learning Workshop*, Tokyo, 1992. ICOT TM-1182.

[20] M. Harao. Analogical reasoning based on higher-order unification. In *Proceedings of the First International Conference on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha.

[21] D. Haussler. Applying Valiant's learning framework to AI concept-learning problems. In Y. Kodratoff and R. Michalski, editors, *Machine learning: an artificial intelligence approach*, volume 3, pages 641–669. Morgan Kaufman, San Mateo, CA, 1990.

[22] P. Idestam-Almquist. Generalization under implication: Expansion of clauses for linear roots. Technical report, Dept. of Computer and Systems Sciences, Stockholm University, 1992.

[23] J-U Kietz. Some lower bounds for the computational complexity of inductive logic programming. In *Proceedings of the European Conference on Machine Learning*, Berlin, 1993. Springer-Verlag.

[24] 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, London, 1992.

[25] R. King, S. Muggleton R. Lewis, and M. Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 89(23), 1992.

[26] R. Kowalski. Logic Programming in Artificial Intelligence. In *IJCAI-91: proceedings of the twelfth international joint conference on artificial intelligence*, pages 596–603, San Mateo, CA, 1991. Morgan-Kaufmann.

[27] P. Langley, G.L Bradshaw, and H. Simon. Rediscovering chemistry with the Bacon system. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 307–330. Tioga, Palo Alto, CA, 1983.

[28] S. Lapointe and S. Matwin. Sub-unification: a tool for efficient induction of recursive programs. In *Proceedings of the Ninth International Machine Learning Conference*, Los Altos, 1992. Morgan Kaufmann.

[29] D.B. Lenat. On automated scientific theory formation: a case study using the AM program. In J.E. Hayes and D. Michie, editors, *Machine Intelligence 9*. Horwood, New York, 1981.

[30] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1984.

[31] A. Kakas P. Mancarella. Generalized stable models: a semantics for abduction. In L. Aiello, E. Sandewall, G. Hagert, and B. Gustavsson, editors, *ECAI-90: proceedings of the ninth European conference on artificial intelligence*, pages 385–391, London, 1990. Pitman.

[32] R. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The AQ15 inductive learning system: an overview and experiments. In *Proceedings of IMAL 1986*, Orsay, 1986. Université de Paris-Sud.

[33] T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[34] F. Mizoguchi and H. Ohwada. Constraint-directed generalization for learning spatial relations. In *Proceedings of the Second Inductive Learning Workshop*, Tokyo, 1992. ICOT TM-1182.

[35] E. Morales. Learning chess patterns. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, London, 1992.

[36] S. Muggleton. A strategy for constructing new predicates in first order logic. In *Proceedings of the Third European Working Session on Learning*, pages 123–130. Pitman, 1988.

[37] S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295–318, 1991.

[38] S. Muggleton. *Inductive Logic Programming*. Academic Press, 1992.

[39] S. Muggleton. Inverting implication. *Artificial Intelligence Journal*, 1993. (to appear).

[40] S. Muggleton. Predicate invention and utility. *Journal of Experimental and Theoretical Artificial Intelligence*, 1993. (to appear).

[41] S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, London, 1992. Academic Press.

[42] S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657, 1992.

[43] S. Muggleton, A. Srinivasan, and M. Bain. Compression, significance and accuracy. In *Proceedings of the Ninth International Machine Learning Conference*, San Mateo, CA, 1992. Morgan-Kaufmann.

[44] S.H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

[45] D. Page and A. Frisch. Generalization and learnability: A study of constrained atoms. In S. Muggleton, editor, *Inductive Logic Programming.* Academic Press, London, 1992.

[46] M. Pazzani, C. Brunk, and G. Silverstein. An information-based approach to integrating empirical and explanation-based learning. In S. Muggleton, editor, *Inductive Logic Programming.* Academic Press, London, 1992.

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

[48] J.R. Quinlan. Generating production rules from decision trees. In *Proceedings of the Tenth International Conference on Artificial Intelligence*, pages 304–307, San Mateo, CA:, 1987. Morgan-Kaufmann.

[49] J.R. Quinlan. Determinate literals in inductive logic programming. In *IJCAI-91: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 746–750, San Mateo, CA:, 1991. Morgan-Kaufmann.

[50] R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[51] B. Richards. *An operator-based approach to first-order theory revision.* PhD thesis, University of Austin, Texas, 1992.

[52] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S. Muggleton, editor, *Inductive Logic Programming.* Academic Press, London, 1992.

[53] S. Russell. Tree-structured bias. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, San Mateo, CA, 1988. Morgan-Kaufmann.

[54] C. Sammut and R.B Banerji. Learning concepts by asking questions. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach. Vol. 2*, pages 167–192. Morgan-Kaufmann, San Mateo, CA, 1986.

[55] E.Y. Shapiro. *Algorithmic program debugging.* MIT Press, 1983.

[56] A. Srinivasan, S. Muggleton, and M. Bain. Distinguishing exceptions from noise in non-monotonic learning. In S. Muggleton, editor, *Proceedings of the Second Inductive Logic Programming Workshop*. ICOT TM-1182, Tokyo, 1992.

[57] M. Sternberg, R. Lewis, R. King, and S. Muggleton. Modelling the structure and function of enzymes by machine learning. *Proceedings of the Royal Society of Chemistry: Faraday Discussions*, 93:269–280, 1992.

[58] M. Stickel. A Prolog technology theorem prover: implementation by an extended prolog compiler. *Journal of Automated Reasoning*, 4(4):353–380, 1988.

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

[60] R. Wirth. Learning by failure to prove. In *EWSL-88*, pages 237–251, London, 1988. Pitman.

[61] R. Wirth and P. O'Rorke. Constraints for predicate invention. In S. Muggleton, editor, *Inductive Logic Programming*, London, 1992. Academic Press.