An Algorithm for Learning Abductive Rules

Evelina Lamma, Michela Milano, Fabrizio Riguzzi
DEIS, Università di Bologna, Viale Risorgimento 2
I-40136 Bologna, Italy, Tel. +39 51 6443033, Fax. +39 51 6443073
{elamma,mmilano,friguzzi}@deis.unibo.it

Paola Mello
Dip. di Ingegneria, Università di Ferrara,
Via Saragat 1, 41100 Ferrara, Italy
pmello@ing.unife.it

Abstract

We propose an algorithm for learning abductive logic programs from examples. We consider the Abductive Concept Learning framework, an extension of the Inductive Logic Programming framework in which both the background and the target theories are abductive logic programs and the coverage of examples is replaced by abductive coverage. The two main benefits of this integration are the increased expressive power of the background and target theories and the possibility of learning in presence of incomplete knowledge. We show that the algorithm is able to learn abductive rules and we present an application of the algorithm to a learning problem in which the background knowledge is incomplete.

Keywords: Abductive Logic Programming. Inductive Logic Programming

1 Introduction

Inductive Logic Programming (ILP) [4] is a research area covering the intersection of Machine Learning and Logic Programming. Its aim is to devise systems that are able to learn logic programs from examples and a background knowledge. Recently, this research area has gained a wider attention as more and more systems are being devised that effectively perform the learning task [15]. As the target of real world applications is becoming nearer, new problems have started to arise, in particular regarding the quality of the data that is typically available. Most of the times the set of training examples and the background knowledge will not be known with certainty because the available data can be affected by noise and/or be incomplete. In this paper we concentrate on the case in which the background knowledge is incomplete. Up to now, very few systems have been developed that are able to learn also in the presence of incomplete knowledge [17, 15, 9].

In order to overcome the lack of information, we integrate induction with the hypothetical reasoning of abduction [16]. Abduction is generally understood as reasoning from effects to causes or explanations. Given a theory T and a formula G, the goal of abduction is to find a set of atoms Δ (explanation) that, together with T, entails G and that is consistent with a set of integrity constraints IC. The atoms in Δ are *abduced*: they are assumed true in order to prove the goal. In the integrated learning framework, when a fact that is necessary for deriving a positive example is missing, we assume it, provided that is consistent with integrity constraints. In this way, we are able to cover the example even if the background knowledge is incomplete.

We consider the Abductive Concept Learning (ACL) framework, proposed by Dimopoulos and Kakas [6], that is an extension of the ILP framework in which both the background knowledge B and the target program P are abductive logic programs. The target program can contain new rules and new integrity constraints. In this extended framework, the coverage of examples is not done through deductive entailment but through abductive entailment. We propose an algorithm that is able to learn an abductive logic program containing only new rules. It is an extension of a basic top-down algorithm adopted in ILP [4]. The extended algorithm takes into account abducibles and integrity constraints, and is intertwined with the proof procedure defined in [11] for abductive logic programs. The key idea is that the abductive proof procedure is used for the coverage process of positive and negative examples in substitution of the deductive proof procedure of logic programming.

The paper is organized as follows: in section 2 we recall the main concepts of ALP, ILP and the definition of the abductive learning framework. In section 3 we present the algorithm for learning abductive rules. We then show in section 4 an example of the application of the algorithm. Related works are presented in section 5. In section 6 we conclude and present the directions for future works.

2 Abductive and Inductive Logic Programming

2.1 Abductive Logic Programming

In the context of abduction, missing information is represented by (user-defined) abducible predicates, possibly constrained by integrity constraints. An *abductive logic program* [10] is a triple $\langle P, \mathcal{A}, IC\rangle$ where:

- P is a normal logic program;
- $\mathcal{A}$ is a set of *abducible predicates*
- IC is a set of integrity constraints in the form of denials, i.e.:

  $\leftarrow A_1, \ldots, A_m, not\_A_{m+1}, \ldots, not\_A_{m+n}.$

Negation as failure is replaced, in ALP, by negation by default and is obtained in this way: for each predicate symbol p, a new predicate symbol not_p is added to the set $\mathcal{A}$ and the integrity constraint $\leftarrow p(\vec{X}), not\_p(\vec{X})$ is added to IC, where $\vec{X}$ is a tuple of variables.

Given an abductive program $\langle P, \mathcal{A}, IC\rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of ground atoms Δ (*abductive explanation*) of predicates in $\mathcal{A}$ which together with P entails G, i.e. $P \cup \Delta \models G$. It is also required that the program $P \cup \Delta$ is consistent with respect to IC, i.e. $P \cup \Delta \models IC$.

In [11] a proof procedure for abductive logic programs has been defined. This procedure starts from a goal and results in a set of consistent hypothesis (abduced literals) that together with the program allow to derive the goal. We have extended this proof procedure in order to allow for abducible predicates to have a partial definition. Some rules may be available for them and we can make assumptions about missing facts.

2.2 Inductive Logic Programming

The ILP problem can be defined as [4]:

Given:

a set $\mathcal{P}$ of possible programs
a set E^+ of positive examples
a set E^- of negative examples
a consistent logic program B such that
$B \not\vdash e^+$ for at least one $e^+ \in E^+$.

Find:

a logic program $P \in \mathcal{P}$ such that
$\forall e^+ \in E^+,\ B \cup P \vdash e^+$ (*completeness*)
$\forall e^- \in E^-,\ B \cup P \not\vdash e^-$ (*consistency*).

Let us introduce some terminology. The sets E^+ and E^- are called *training sets*. The program P that we want to learn is the *target program* and the predicates which are defined in it are *target predicates*. The program B is called *background knowledge* and contains the definitions of the predicates that are already known. We say that the program P *covers* an example e if $P \cup B \vdash e$. Therefore the conditions that the program P must satisfy in order to be a solution to the ILP problem can be expressed as "P must cover all positive examples and must not cover any negative examples". The set $\mathcal{P}$ is called the *hypothesis space*. The importance of this set lies in the fact that it defines the search space of the ILP system. In order to be able to effectively learn a program, this space must be restricted as much as possible. If the space is too big, the search could result infeasible.

The *language bias* (or simply *bias* in this paper) is a description of the hypothesis space. Many formalisms have been introduced in order to describe this space [4], we will consider only a very simple bias in the form of a set of literals which are allowed in the body of the clauses for the target predicates.

2.3 Abductive Concept Learning

In [6] the authors define a new learning problem called Abductive Concept Learning that consists in learning an abductive logic program starting from a set of positive and negative examples and an abductive background theory.

Given

a set $\mathcal{P}$ of possible abductive programs
a set of positive examples E^+,
a set of negative examples E^-,
an abductive theory $AT = \langle T, A, IC \rangle$ as background theory.

Find

A new abductive theory $AT' = \langle T', A, IC' \rangle \in \mathcal{P}$ such that
$\forall e^+ \in E^+,\ AT' \vdash_A e^+$,
$\forall e^- \in E^-,\ AT' \not\vdash_A e^-$.

where $AT' \vdash_A e$ means that e is abductively provable from AT', i.e., there exist an abductive explanation for e from AT'. We say that AT' *abductively covers* e.

The abductive program that is learned can contain new rules (eventually containing abducibles in the body) and new integrity constraints, but not new abducible predicates. This is different from the framework in [8] where also new abducibles can be introduced.

3 An algorithm for Learning Abductive Rules

We present an algorithm that is able to learn abductive rules. The algorithm is obtained from the basic top-down ILP algorithm [4], by substituting the usual notion of coverage of examples with the notion of *abductive coverage*.

The learned rules can contain abducible literals in their bodies and positive examples are tested by starting an abductive derivation for each of them ($\leftarrow e^+$), while negative examples are tested by starting an abductive derivation for the negation of each of them ($\leftarrow not\_e^-$). We will employ the abductive proof procedure defined in [11], with a few modifications for allowing abducibles to have partial definitions in the program (i.e. rules for the abducibles).

Testing the negation of each negative example is different from testing that $AT \not\vdash_A e^-$. Even if $AT \vdash_A not\_e^-$, it may still be possible that there exist an abductive explanation for e^- in AT. For this reason, after the rules have been learned, the assumptions made in the derivation of negative examples are added to the theory, in the form of facts for positive literals and of integrity constraints for negative ones. In this way, we ensure that for each e^-, $AT' \not\vdash_A e^-$. We decided to add to the target theory also the abducibles used to cover positive examples, even if they are not necessary to preserve correctness. In this way, we not only learn a general theory, but we also complete the background theory.

We have increased the ways in which a positive example can be covered and a negative example ruled out. A positive example can be covered without abducing anything, thus expressing the fact that the example is surely positive, or it can be covered by making some assumptions, thus expressing the fact that we do not have complete confidence in its coverage, but that it is consistent with our current representation of the domain. Similarly, a negative example can be ruled out with certainty, when its negation is derived without abducing anything, or it can be ruled out under certain assumptions. The learning power of the algorithm is therefore greater than that of an ILP system because it is able to learn even when the knowledge about the domain is not completely specified, as it is often the case for real learning problems, by making hypothesis about the unknown parts of the domain, provided that these are consistent with known integrity constraints.

The basic top-down inductive algorithm [4] learns programs by generating clauses one after the other and generates clauses by means of specialization. The basic inductive algorithm is constituted by two nested loops: the covering loop (figure 1) and the specialization loop (figure 2). At each iteration of the covering loop a new clause is generated such that it covers at least one positive example and no negative one. The positive examples covered by the rule are removed from the training set. The algorithm ends when the training set becomes empty. A clause is generated in the specialization loop: we start with a clause with an empty body, and we add a literal to the body until the clause does not cover any negative example while still covering at least one positive. The basic top-down algorithm is extended in the following respects.

First, in order to determine the positive examples E^+_{Rule} covered by the generated rule $Rule$ (procedure TestCoverage in figure 3), an abductive derivation is started for each positive example. This derivation results in a (possibly empty) set of consistent hypotheses (abduced literals). We give as input to the abductive procedure also the set of literals abduced in the derivation of the previously covered examples. In this way, we ensure that the assumptions made during the derivation of the current example are consistent with the assumptions previously raised for covering other examples.

Second, in order to check that no negative example is covered by the generated rule

```
procedure LearnAbdRules(
    inputs : $E^+, E^-$ : training sets,
        $AT = \langle T, A, IC \rangle$ : background abductive theory,
    outputs : $H$ : learned theory, $\Delta$ : abduced literals)

$H := \emptyset$
$\Delta := \emptyset$
while $E^+ \neq \emptyset$ do
    GenerateRule(input: $AT, H, E^+, E^-, \Delta$; output: $Rule, E^+_{Rule}, \Delta'$)
    $E^+ := E^+ - E^+_{Rule}$
    $H := H \cup \{Rule\}$
    $\Delta := \Delta'$
endwhile
Add $\Delta$ to the hypothesis $H$:
    positive literals $p(X)$ as facts $p(X)$.
    negative literals $not\_p(X)$ as constraints $\leftarrow p(X)$.
output $H$
```

Figure 1: The covering loop

```
procedure GenerateRule(
    inputs : $AT$ : background abductive theory, $H$ : current hypothesis,
        $E^+, E^-$ : training sets, $\Delta$ : current set of abduced literals
    outputs : $Rule$ : rule, $E^+_{Rule}$ : positive examples covered by $Rule$,
        $\Delta'$ : new set of abduced literals

Select a predicate to be learned $p$
Let $Rule = p(X) \leftarrow true$.
TestCoverage(input: $Rule, AT, H, E^+, E^-, \Delta$,
    output: $E^+_{Rule}, E^-_{Rule}, \Delta'$)
While $E^-_{Rule} \neq \emptyset$ do
    Select a literal $L$ from the language bias
    Add $L$ to the body of $Rule$
    TestCoverage(input: $Rule, AT, H, E^+, E^-, \Delta$,
        output: $E^+_{Rule}, E^-_{Rule}, \Delta'$)
    if $E^+_{Rule} = \emptyset$
        backtrack to a different choice for $L$
endwhile
output $Rule, E^+_{Rule}, \Delta'$
```

Figure 2: The specialization loop

```
procedure TestCoverage(
    inputs : $Rule$ : rule, $AT$ : background abductive theory,
        $H$ : current hypothesis, $E^+, E^-$ : training sets,
        $\Delta$ : current set of abduced literals
        outputs: $E^+_{Rule}, E^-_{Rule}$: positive and negative
        examples covered by $Rule$
        $\Delta'$ : new set of abduced literals

$E^+_{Rule} = E^-_{Rule} = \emptyset$
$\Delta_{in} = \Delta$
for each $e^+ \in E^+$ do
    if AbductiveDerivation($e^+, \langle T \cup H \cup \{Rule\}, A, IC\rangle, \Delta_{in}, \Delta_{out}$)
            succeeds then
        Add $e^+$ to $E^+_{Rule}$
        $\Delta_{in} = \Delta_{out}$
    endif
endfor
for each $e^- \in E^-$ do
    if AbductiveDerivation($not\_e^-, \langle T \cup H \cup \{Rule\}, A, IC\rangle, \Delta_{in}, \Delta_{out}$)
            succeeds then $\Delta_{in} = \Delta_{out}$
    else
        Add $e^-$ to $E^-_{Rule}$
    endif
endfor
$\Delta' = \Delta$
output $E^+_{Rule}, E^-_{Rule}, \Delta'$
```

Figure 3: Coverage testing

Rule, an abductive derivation is started for the negation of each negative example (figure 3). Also in this case, each derivation does not start with an empty set of abducibles but it starts from the set of abducibles previously assumed. Therefore the set of abducibles is passed on from derivation to derivation and gradually extended. This is done across different clauses as well.

Third, at the end of the learning phase, the abduced literals are added to the learned theory, so that no incompatible assumptions can be made when the theory is effectively used. The positive literals are added as facts, while the negative literals $not\_p(X)$ are added as integrity constraints of the form $\leftarrow p(X)$.

The algorithm has been implemented in Prolog using Sicstus Prolog 3#3. The program is composed of five main predicates. `induce` is the predicate that starts the induction process. It initializes the training sets and calls `covering_loop` that implements the covering loop by tail recursion. At each iteration, it generates a new clause, by calling the predicate `generate_cl`, and updates the training set. `generate_cl` selects a predicate to be learned, reads the bias and starts the specialization loop, then returns a new consistent clause. `specialize` implements the specialization loop by tail recursion. It checks the coverage of the current clause, if it still covers some negative examples, it calls `select_literal_to_add` that select the new literal to add from the bias. If the rule does not cover any positive examples, `specialize` fails and backtracking is performed on the choice of the literal previously selected by `select_literal_to_add`.

4 Example

In this section, we show a simple example of the application of the algorithm. Let us consider the case of an abductive background theory $B = \langle P, \mathcal{A}, IC \rangle$ and training set:

$$\begin{aligned}
P = \quad & \{parent(john, mary), male(john),\\
& parent(david, steve),\\
& parent(kathy, ellen), female(katy)\}\\
A = \quad & \{male, female\}\\
IC = \quad & \{\leftarrow male(X), female(X)\}\\
E^+ = \quad & \{father(john, mary), father(david, steve)\}\\
E^- = \quad & \{father(katy, ellen), father(john, steve)\}
\end{aligned}$$

Moreover, let the bias be

$$\begin{aligned}
father(X,Y) \leftarrow \alpha \text{ where } \alpha \subset \{& parent(X,Y), parent(Y,X),\\
& male(X), male(Y), female(X), female(Y)\}
\end{aligned}$$

In this case, the algorithm learns the rule

$$father(X,Y) \leftarrow parent(X,Y), male(X).$$

making the assumptions $\Delta = \{male(david), not\_female(david), not\_male(katy)\}$.

Let us describe now in detail the behaviour of the algorithm. At the first iteration of the specialization loop, the algorithm generates the rule

$$father(X,Y) \leftarrow .$$

which covers all the positive examples but also covers all the negative ones. Therefore another iteration is started and the literal $parent(X,Y)$ is added to the rule

$$father(X,Y) \leftarrow parent(X,Y).$$

This clause also covers all the positive examples but also the negative example
$father(katy, ellen)$.
Note that up to this point no abducible literal has been added to the rule, therefore no abduction has been made and the set Δ is still empty. Now, an abducible literal is added to the rule, $male(X)$, obtaining

$$father(X,Y) \leftarrow parent(X,Y), male(X).$$

At this point the coverage of examples is tested. $father(john, mary)$ is covered without abducing anything because we have the fact $male(john)$ in the background. The other positive example, $father(david, steve)$, is covered with the abduction of $male(david)$, $not\_female(david)$ and the Δ set becomes $\{male(david), not\_female(david)\}$. Then the covered negative example is tested by starting an abductive derivation for

$$\leftarrow not\_father(katy, ellen).$$

This derivation succeeds abducing $not\_male(katy)$ which is consistent with the fact $female(katy)$ and the constraint $\leftarrow male(X), female(X)$. Now, no negative example is covered, therefore the specialization loop ends. The positive examples covered by the rules are removed from the training set which becomes empty. Therefore also the covering loop terminates and the algorithm ends by adding the literals in Δ to the target program. $male(david)$ is added as a fact, while $not\_male(katy)$ and $not\_female(david)$ are added as the integrity constraints $\leftarrow male(katy)$ and $\leftarrow female(david)$.

5 Related Work

We start by mentioning our previous works in the field and then we describe related works by other authors.

In [8] we show how, by integrating induction with abduction, we can learn exceptions to rules, learn from integrity constraints and learn binary constraints. Differently from the present paper, the algorithm in [8] can introduce new abducibles in order to cope with exceptions. Moreover, it does not add the abduced literals to the target programs, thus not fully supporting ACL.

In [14] we have proposed an algorithm for learning abductive rules obtained modifying the extensional ILP system FOIL [17]. Extensional systems differ from intensional ones (as the one presented in this paper) because they employ a different notion of coverage, namely *extensional coverage*. When testing a rule with extensional coverage, the literals in the body of the rule are proved true either if they are derived in the background knowledge or if they belong to the training set. The partial theory learned so far is not used for the derivation of examples because the training set is used for the definition of the target predicates: this has the advantage of allowing the system to learn clauses independently from each other, avoiding the need of considering different orders in learning the clauses and the need for backtracking on clause addition. However, it has a number of disadvantages [19]. In [14] we show how the integration of abduction and induction can solve some of the problems of extensional systems when dealing with recursive predicates and programs with negation.

As concern the integration of abduction and induction, a notable work is that by Dimopoulos and Kakas [6]. In this paper, the authors suggest two approaches for the integration of abduction in learning. In the first, abduction is used in a preliminary stage to explain the training data of a learning problem in order to generate suitable or relevant background data on which to base the successive inductive generalization process. The second approach is ACL, that we have adopted in this paper. The main advantage of the integration in [6] is, as in our framework, that it allows to possibly learn rules in presence of missing information, and later classify new examples that may be incompletely described. Differently from us, the framework in [6] allows the generation of integrity constraints for specializing a rule, while we allow only the addition of a literal to the body of the clause. Adding integrity constraints for specializing the rules means that each atom derived by using the rules must be checked against the constraints, which can be computationally very expensive.

In [13] an algorithm for learning abductive theories with general integrity constraints is proposed. The algorithm learns the theory in two steps: first the rules are learned, using an algorithm called PACL which is similar to LearnAbdRules, and then the constraints are learned, using the system ICL [20]. However, the approach for learning integrity constraints is still in an early stage of development and must be further investigated.

In [2] a parametric framework is proposed that can be instantiated to both abduction and induction. This framework is used in order to transform a proof procedure for abduction, namely SLDNFA, in a proof procedure for induction, called SLDNFAI. Informally, SLDNFA is extended in order to be able to abduce not only ground facts but also rules. However, the authors obtain a learning framework which is equivalent to the ILP one, while we obtain a more powerful framework where we can learn in presence of incomplete knowledge because abduction helps the induction process by making assumptions on missing data.

Another related work is reported in [1], where the authors present a system, called RUTH, for theory revision based on ILP. RUTH is able to cope with definite, functor-free clauses, and integrates intensional database updating with incremental concept-learning. Apart from adding and deleting clauses and facts, in [1] the authors also employ an abductive operator which allows RUTH to introduce missing factual knowledge into the knowledge base. Added (and possibly violated) integrity constraints correspond to positive (uncovered) and negative (covered) examples. In order to handle (uncovered) positive examples, theory revision recovers from the arisen inconsistency by either (*i*) adding an example as a fact in the database, or (*ii*) building a maximally general clause that covers the example, or (*iii*) abducing one or more new facts. In order to handle (covered) negative examples, theory revision recovers from the arisen inconsistency by deleting one of the clauses that contribute to the SLDNF proof of the example. Both RUTH and our framework can treat as abducibles *some* of the program predicates. Differently from [1], we avoid clause retraction, and rather prefer to specialize the clauses in order to rule out negative examples. In this respect, we do not fully support theory revision.

6 Conclusions and Future Work

We have presented an algorithm for performing Abductive Concept Learning. In this extended framework, the ILP problem is modified by considering both the background and target theories as abductive theories and by replacing the notion of coverage with that of abductive coverage. The algorithm is able to learn abductive theories containing new rules. It is obtained from the basic top-down algorithm of ILP by substituting the coverage of examples using resolution with the coverage using an abductive proof procedure.

We have implemented in Prolog the algorithm proposed and, in the future, we will test the algorithm on real domains in which the incompleteness of the data causes problem to usual ILP systems. As regards the theoretical aspects, we will further investigate the problem of extending the algorithm for learning full abductive theories, comprehending also integrity constraints. The integration of the algorithm with other systems for learning constraints, proposed in [13], seems very promising and more work is needed to reach the objective of a system for learning full abductive theories.

Acknowledgments

This research was partially funded by the MURST Project 40% "Rappresentazione della conoscenza e meccanismi di ragionamento". Part of this work was done during two visits of Fabrizio Riguzzi to the University of Cyprus, which were funded by the ESPRIT IV Long Term Research Project Number 20237 ILP^2. He would like to thank Antonis Kakas for making these visits possible and for many interesting discussions on the topic of this paper.

References

[1] H. Adé and M. Denecker. RUTH: An ILP Theory Revision System. In *Proceedings ISMIS94*, 1994.

[2] H. Adé and M. Denecker. AILP: Abductive Inductive Logic Programming. In *Proceedings IJCAI95*, 1995.

[3] M. Bain and S. Muggleton. *Non-Monotonic Learning*, chapter 7. Academic Press, 1992.

[4] F. Bergadano and D. Gunetti. *Inductive Logic Programming.* MIT press, 1996.

[5] Y. Dimopoulos and A. Kakas. Learning Non-monotonic Logic Programs: Learning Exceptions. In *Proceedings ECML95*, 1995.

[6] Y. Dimopoulos and A. Kakas. Abduction and Learning. In *Advances in Inductive Logic Programming.* IOS Press, 1996.

[7] K. Eshghi and R.A. Kowalski. Abduction compared with Negation by Failure. In *Proceedings of ICLP89*, 1989.

[8] F. Esposito, E. Lamma, D. Malerba, P. Mello, M.Milano, F. Riguzzi, and G. Semeraro. Learning Abductive Logic Programs. In *Proceedings of the ECAI96 Workshop on Abductive and Inductive Reasoning*, 1996.

[9] N. Inuzuka, M. Kamo, N. Ishii, H. Seki, and H. Itoh. Top-down induction of logic programs from incomplete samples. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 119–136. Stockholm University, Royal Institute of Technology, 1996.

[10] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2:719–770, 1993.

[11] A.C. Kakas and P. Mancarella. On the relation between Truth Maintenance and Abduction. In *Proceedings of PRICAI90*, 1990.

[12] A.C. Kakas, P. Mancarella, and P.M. Dung. The acceptability semantics for logic programs. In *Proceedings of ICLP94*, 1994.

[13] A.C. Kakas and F. Riguzzi. Abductive Concept Learning. Technical Report TR-96-15, University of Cyprus, Computer Science Department, 1996.

[14] E. Lamma, P. Mello, M. Milano, and F. Riguzzi. Introducing Abduction into (Extensional) Inductive Logic Programming Systems. submitted.

[15] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.

[16] D.L. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 32, 1988.

[17] J. R. Quinlan and R.M. Cameron-Jones. Induction of Logic Programs: FOIL and Related Systems. *New Generation Computing*, 13:287–312, 1995.

[18] L. De Raedt and M. Bruynooghe. On negation and three-valued logic in interactive concept learning. In *Proceedings of ECAI90*, 1990.

[19] L. De Raedt, N. Lavrač, and S. Džeroski. Multiple Predicate Learning. In *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, 1993.

[20] L. De Raedt and W. Van Lear. Inductive Constraint Logic. In *Proceedings of the 5th International Workshop on Algorithmic Learning Theory*, 1995.