Introducing Abduction into (Extensional) Inductive Logic Programming Systems

E. Lamma[1], P. Mello[2], M. Milano[1], F.Riguzzi[1]

[1] DEIS, Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{elamma,mmilano,friguzzi}@deis.unibo.it
Tel.+39 51 6443033, Fax. +39 51 6443073
[2] Istituto di Ingegneria, Università di Ferrara
Via Saragat, 41100 Ferrara, Italy
pmello@ing.unife.it

Abstract. We propose an approach for the integration of abduction and induction in Logic Programming. In particular, we show how it is possible to learn an abductive logic program starting from an abductive background knowledge and a set of examples. By integrating Inductive Logic Programming with Abductive Logic Programming we can learn in presence of incomplete knowledge. Incomplete knowledge is handled by designating some pieces of information as abducibles, that is, possible hypotheses which can be assumed, provided that they are consistent with the current knowledge base. We then specialize the framework for FOIL, an ILP system adopting extensional coverage. In particular, we propose an extension of the FOIL algorithm that is able to learn from incomplete data.

Content Areas: Machine Learning, Nonmonotonic reasoning

1 Introduction

Both abduction and induction have been recognized as powerful mechanisms for hypothetical reasoning in the presence of incomplete knowledge [8, 12, 13]. Abduction is generally understood as reasoning from effects to causes or explanations, and captures also other important issues such as reasoning in presence of incomplete information and reasoning with defaults and beliefs (see for instance [14]). Incomplete knowledge is handled by designating some pieces of information as abducibles, that is possible hypotheses which can be assumed, provided that they are consistent with the current knowledge base.

Integrating induction and abduction makes it possible to learn in the presence of incomplete information on the background or target relations. This is the case when there are undefined ground atoms for the relations possibly constrained by integrity constraints.

The main contributions of the paper are the following:

- to present a unifying framework for the integration of Inductive Logic Programming (ILP, for short [3]) and Abductive Logic Programming (ALP, for short in the following [8]). In particular, we present a general approach where it is possible to learn an abductive logic program starting from an abductive background knowledge.
- to specialize the framework for an extensional ILP system, FOIL [15]. In particular, we present an *Extentional Abductive Proof Procedure* for checking the coverage of positive and negative examples.

The following simplified example shows how we would like to learn a program for performing fault diagnosis from a set of (incomplete) bicycle faults.

Example 1

Let us consider the case of a background knowledge containing the following clauses, where the last one represents a constraint:

$flat\_tyre(bike_1) \leftarrow .$
$tyre\_holds\_air(bike_3) \leftarrow .$
$tyre\_holds\_air(bike_4) \leftarrow .$
$\leftarrow flat\_tyre(X), tyre\_holds\_air(X).$

and let the set of abducible predicates be $\{flat\_tyre, broken\_spokes\}$. We would like to learn relation $wobbly\_wheel$, when the training set is constituted by:

$E^+ = \{wobbly\_wheel(bike_1), wobbly\_wheel(bike_2),$
$wobbly\_wheel(bike_4)\}$
$E^- = \{wobbly\_wheel(bike_3)\}$

By integrating abduction and induction, we would like to produce the clauses:

$wobbly\_wheel(X) \leftarrow flat\_tyre(X).$
$wobbly\_wheel(X) \leftarrow broken\_spokes(X).$

by assuming the following set of hypotheses:

$\{flat\_tyre(bike_2), broken\_spokes(bike_4),$
$not\ flat\_tyre(bike_3), not\ broken\_spokes(bike_3)\}.$

A standard ILP system would be able to learn the clauses above only when the user also specifies the abduced hypotheses in the background knowledge. Nonetheless, in many cases one cannot define a complete background knowledge, namely all the information about predicates $flat\_tyre$ and $broken\_spokes$, but only gives some hints about them (e.g., their relationships with other predicates through constraints).

In the paper, we first extend a basic top-down ILP algorithm with abduction in order to obtain the behavior informally explained through the example above. We then specialize this framework for FOIL, an ILP system adopting the notion of extensional coverage.

2 Integrating Induction and Abduction

In the following, we use the basic concepts and terminology of logic programming. We limit the language to have no function symbol in order to have a finite Herbrand Universe.

2.1 Abductive Logic Programming

We recall now the definition of abductive logic program.

An *abductive logic program* is a triple $\langle P, \mathcal{A}, IC \rangle$ where:

- P is a normal logic program, i.e., a set of clauses of the form
  $A_0 \leftarrow A_1, \ldots, A_m, \; not \; A_{m+1}, \ldots, not \; A_{m+n}$
  where $m, n \geq 0$ and each A_i $(i = 0, \ldots, m+n)$ is an atom;
- $\mathcal{A}$ is a set of *abducible predicates*;
- IC is a set of *integrity constraints* (denials, for simplicity) of the form
  $\leftarrow A_1, \ldots, A_m, \; not \; A_{m+1}, \ldots, not \; A_{m+n}$

Abduction can be used to extend standard SLD-resolution in order to generate *conditional answers* for a given goal. Such answers are (possibly minimal) sets of *consistent* abducibles.

Negation as Failure is replaced, in ALP, by Negation by Default [6] 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 $\leftarrow p(\mathbf{X}), not\_p(\mathbf{X})$ is added to IC, where $\mathbf{X}$ is a tuple of variables.

In the following, we adopt the three-valued semantics for abductive logic programs defined in [4]. Thus, we use a three-valued logic in which an atom can be *true*, *false* or *unknown*.

Operationally, we rely on the proof procedure defined by Kakas and Mancarella [9]. The proof procedure uses the notion of *abductive* and *consistency derivation*. Intuitively, an abductive derivation is the usual LP derivation suitably extended in order to consider abducibles. As soon as an abducible atom δ is encountered, it is added to the current set of hypotheses, and it must be proved that any integrity constraint containing δ fails via a consistency derivation. During this latter procedure, when an abducible is encountered, in order to prove its failure, an abductive derivation for its complement is attempted.

2.2 Inductive Logic Programming with Abduction

The ILP problem can be defined as [3]:

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 *backgroung knowledge*

Find:

a logic program $P \in \mathcal{P}$ such that
$\forall e^+ \in E^+, \; B \cup P \models e^+ \;\; (P \; covers \; e^+)$
$\forall e^- \in E^-, \; B \cup P \not\models e^- \;\; (P \; does \; not \; cover \; e^-)$.

In order to integrate abduction and induction, we consider a modified version of the ILP problem where both the background knowledge and the learned

program are abductive logic program. Moreover, coverage of examples through entailment is replaced by coverage through abductive entailment:

$\forall e^+ \in E^+, B \cup P \models_A e^+$ (P *covers* e^+)

$\forall e^- \in E^-, B \cup P \models_A not\_e^-$ (P *does not cover* e^-).

where $B \cup P \models_A e^+$ means that there exist at least one abductive explanation for e^+ in $B \cup P$.

The training sets, E^+ and E^-, define possibly partial information for the learning process, and both positive and negative examples can be enlarged by the addition of some positive or negative abducible atoms provided that they are consistent with the background knowledge, and integrity constraints in particular.

The basic top-down inductive algorithm [3] learns programs by generating clauses one after the other, and generates clauses by means of specialization. Let T denote the set of induced clauses, initially empty, and E^+ and E^- be the training sets (positive and negative examples, respectively). The basic inductive algorithm is sketched in figure 1.

while some positive examples in E^+
    are not covered by a clause in T **do**
  Generate one clause C
  Remove from E^+ positive examples *covered* by C
  Add C to T

Genarate one Clause C:
Select a predicate P that must be learned
Set clause C to be $p(\mathbf{X}) \leftarrow$.
while C covers some negative example **do**
  Select a literal L from the language bias
  Add L to the antecedent of C
  if C does not cover any positive example
    then backtrack to different choices for L
return C (or fail if backtracking
    exhausts all choices for L)

Fig. 1. Basic ILP algorithm

This basic inductive algorithm is extended with abduction into the following respects:

- First, the generated clauses can contain abducible predicates in their body, in analogy with the frameowrk in [5, 7].
- Second, in order to determine the positive examples covered by the generated clause C, and to be removed, an *abductive* derivation is started for each of them. As well, in order to check that no negative example is covered by the generated clause C, an *abductive* derivation is started for the complement

of each negative example. In both cases, this is achieved by exploiting the abductive proof procedure defined in [9] or by extending it with the notion of extensional coverage, as presented in section 4, for the extensional ILP systems. During the abductive procedure, some abducibles can be assumed true or false. Note that literals abduced during the abductive derivation have to be recorded to avoid inconsistent future assumptions.

This behavior is specialized in section 4 for the FOIL system.

3 Overview of FOIL

FOIL [15] is an *extensional* ILP system. In FOIL, both target and background relations are described extensionally by sets of *tuples* of constants: the set of *positive* (respectively *negative*) tuples for a relation contains the tuples that belong (do not belong) to the relation. In the following we will refer to such tuples as $\oplus$ and $\ominus$ respectively. The learned definition is a logic program that *covers* all the $\oplus$ tuples and none of the $\ominus$ tuples of the target relation(s). In order to test if a tuple is covered, FOIL uses the notion of *extensional coverage* [11]:

Definition 1: Let $\mathcal{P}$ be the program defining the target predicate, let E^+ be the set of positive examples of the target predicate, let $\mathcal{M}_{\mathcal{BK}}$ be the model of the background knowledge, and let $e = p(\mathbf{t})$ be an example. $\mathcal{P}$ *extensionally covers* the tuple $\mathbf{t}$ (and the example e) if there exists a ground instance of a clause of $\mathcal{P}$, $l \leftarrow l_1, \ldots, l_n$ such that $l = e$ and for all i, $l_i \in (E^+ \cup \mathcal{M}_{\mathcal{BK}})$.

3.1 The learning algorithm of FOIL

FOIL uses a top-down learning algorithm. It differs from the basic ILP top-down algorithm shown in figure 1 only in the way the clause is specialized (procedure "Generate one clause", figure 2)

Clause specialization is guided by the bindings of the variables in the partial clause that make the body true. If the clause contains k variables, a *binding* is a k-tuple of constants that specifies the values of all variables in order. Each possible binding is labeled $\oplus$ or $\ominus$ depending on whether the tuple of values for the variables in the clause head belongs or does not belong to the target relation.

4 Extending FOIL with Abduction

Each relation is described by a set of $\oplus$ tuples and a set of $\ominus$ tuples. We assume all the tuples not specified by the user as $\oplus$ or $\ominus$ to be unknown ($\odot$ in the following). We would like FOIL to make assumptions about $\odot$ tuples in order to cover positive examples or exclude negative ones. By using abduction, we ensure that the assumptions made are consistent. The $\odot$ tuples for each relation determine the set of abducible ground atoms. We need to give as input to FOIL as well a set of integrity constraints.

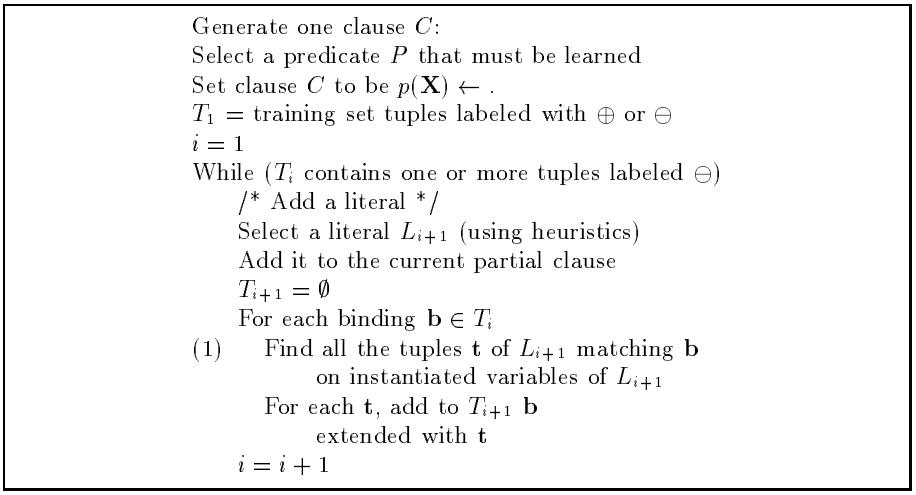

Fig. 2. FOIL: clause specialization algorithm

In addition, we have to modify the notion of extensional coverage for coping with abductive logic programs.

Definition 2: Let $\mathcal{BK}$ be an abductive theory for the background knowledge, $\mathcal{P}$ be the abductive program defining the target predicate. Let E^+ be the set of positive examples of the target predicate, and $e = p(\mathbf{t})$ be an example. $\mathcal{P}$ *extensionally covers* the tuple $\mathbf{t}$ (and the example e) if there exists a model $\mathcal{M}_{\mathcal{BK}}$ under the three valued abductive semantics, a ground instance of a clause of $\mathcal{P}$, $l \leftarrow l_1, \ldots, l_n$ and a set Δ of abducibles atoms, consistent with $IC \cup \mathcal{M}_{\mathcal{BK}}$, such that $l = e$ and for all i, $l_i \in (E^+ \cup \Delta \cup \mathcal{M}_{\mathcal{BK}})$.

When FOIL adds a new literal to a clause, it tests if the tuples covered by the previous clause are covered as well by the new clause. If, for a certain tuple, the new literal is unknown and the corresponding predicate is abducible, we try to abduce it. We use an abductive derivation modified to deal with extensional coverage. If the abductive derivation succeeds, the literal can be assumed true and has to be recorded because future abductive derivations should not derive anything that is inconsistent with them. Therefore, the corresponding tuples are added to the set of $\oplus$ or $\ominus$ tuples for the relations.

More in detail, FOIL algorithm must be modified in point (1) (in figure 2). At that point, the set of bindings T_i is tested. For each binding b in T_i, if it matches with one or more $\oplus$ (resp. $\ominus$ if L_{i+1} is negative) tuples of L_{i+1}, the original tuple for the target relation is covered and b is extended and added to T_{i+1}. If b matches with a $\ominus$ (resp. $\oplus$ if L_{i+1} is negative), it is discarded. Otherwise (b does not match neither with a $\oplus$ nor with a $\ominus$ tuple), b matches with at least an $\odot$ tuple. Therefore, if b belongs to positive examples, we try to cover it by means of abduction. In particular, we pick up one of the matching $\odot$ tuples u of L_{i+1} and assume it true (resp. false if L_{i+1} is negative), provided that this

is consistent with integrity constraints and with other abduced literals. This is done by starting an *extensional abductive derivation* for L_{i+1}. If it succeeds, L_{i+1} can safely be assumed true and this is done by adding u to the set of $\oplus$ tuples of L_{i+1} (or to the $\ominus$ if the literal is negative). Moreover, b is extended with values for (eventual) new variables in u and added to T_{i+1}.

If b belongs to negative examples, we have to do the opposite: u has to be assumed false if L_{i+1} is positive (true if L_{i+1} is negative). Therefore, an abductive derivation is started for $\overline{L_{i+1}}$[3]

If L_{i+1} is a target predicate, we may have already learned a definition for it. In this case we need to check the rules learned against the new tuples eventually added to the extensional definition of L_{i+1} and, possibly, revise the theory. If a learned rule extensionally covers one of the new negative tuples, the rule must be further specialized. We must run the specialization algorithm for each of the learned rules, each time starting with a training set containing only the new positive and negative tuples. The positive tuples that are covered, are not included in the initial training set for the successive rules, while the negative tuples are kept for all the rules.

If some of the new positive tuples are not covered by any rule, then new rules must be learned to cover these tuples.

The algorithm for extensional abductive derivation is a modification of the one for standard abductive derivation defined by Kakas and Mancarella [9]. Let L be a ground literal to be derived, with $L = R(\mathbf{t})$ or $L = not\_R(\mathbf{t})$. We say that L is *true* if $\mathbf{t}$ is among the $\oplus$ tuples of R ($\ominus$ if L is negative). We say that L is *false* if the opposite is true. We say that it is *unknown* if $\mathbf{t}$ is neither among $\oplus$ nor $\ominus$ tuples of R.

Extensional abductive derivation
L can be derived if:

(A1) L is true,
(A2) if L is unknown, $\mathbf{t}$ is temporarily added to the $\oplus$ tuples of R ($\ominus$ if L is negative) and an *extensional consistency derivation* from L to $\{\}$ is attempted.

The derivation fails in the case in which L is false.

Extensional consistency derivation
An extensional consistency derivation from a literal L to F_n is a sequence $L, F_1, F_2, \ldots, F_n$ where:

(Ci) F_1 is the union of all goals of the form $\leftarrow L_1, \ldots, L_k$ obtained by resolving L with the denials in IC, with no such goal being empty, $\leftarrow$.
(Cii) for each $i > 1$, F_i has the form $\{\leftarrow L_1, \ldots, L_k\} \cup F_i'$ and for some $j = 1 \ldots k$, F_{i+1} is obtained according to one of the following rules:

[3]
$$\overline{L_{i+1}} = \begin{cases} not\_p(\mathbf{X}) & \text{if} L_{i+1} = p(\mathbf{X}) \\ p(\mathbf{X}) & \text{if} L_{i+1} = not\_p(\mathbf{X}) \end{cases}$$

(C1) if L_j is true, then $F_{i+1} = \{\leftarrow L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_k\} \cup F'_i$ (the denial is removed because it is verified);
(C2) if L_j is false, $F_{i+1} = F'_i$ (the literal is removed);
(C3) if L_j is unknown, an extensional abductive derivation is started for $\overline{L_j}$. If it succeeds, $F_{i+1} = F'_i$.

The extensional consistency derivation fails if at least one F_i contains an empty goal $\leftarrow$, and succeeds if $F_n = \{\}$. In the case it fails, $\mathbf{t}$ has to be removed from the set of $\oplus$ (resp. $\ominus$) tuples.

Let us consider the example presented in the introduction. The set of $\odot$ tuples for the relation $flat\_tyre$ is $\{\langle bike_2\rangle, \langle bike_3\rangle, \langle bike_4\rangle\}$, and for the relation $broken\_spokes$ is $\{\langle bike_1\rangle, \langle bike_2\rangle, \langle bike_3\rangle, \langle bike_4\rangle\}$. The first generated clause is:

$wobbly\_wheel(X) \leftarrow flat\_tyre(X)$

This clause covers $wobbly\_wheel(bike_1)$ because $flat\_tyre(bike_1)$ is specified in the background knowledge. In order to cover $wobbly\_wheel(bike_2)$, the system considers $flat\_tyre(bike_2)$. Since it is unknown, an abductive derivation is started. The tuple $\langle bike_2\rangle$ is added to the set of $\oplus$ tuples for $flat\_tyre$ and a consistency derivation is started (step (A2)). The literal is resolved with

$\leftarrow flat\_tyre(X), tyre\_holds\_air(X)$.

giving the goal $\leftarrow tyre\_holds\_air(bike_2)$ (step (Ci)). An abductive derivation for $not\ tyre\_holds\_air(bike_2)$ is started (step (C3)). This derivation succeeds abducing $not\ tyre\_holds\_air(bike_2)$.

The example $wobbly\_wheel(bike_4)$, however, cannot be covered: in fact, we cannot assume $flat\_tyre(bike_4)$ since it is inconsistent with the integrity constraint and $tyre\_holds\_air(bike_4)$. The negative example $wobbly\_wheel(bike_3)$ is ruled out by starting an extensional abductive derivation for $not\_flat\_tyre(bike_3)$. The derivation succeeds and $not\_flat\_tyre(bike_3)$ is assumed true. Tuples corresponding to abduced literals are added to $\oplus$ and $\ominus$ tuples. In order to cover $wobbly\_wheel(bike_4)$, the system generates the clause:

$wobbly\_wheel(X) \leftarrow broken\_spokes(X)$

which covers the example by abducing $broken\_spokes(bike_4)$. Similarly to the previous case, the negative example is ruled out by assuming

$not\_broken\_spokes(bike_3)$.

5 Related Work

The relationship between abduction and learning has been studied recently by several authors. In general, the question of how abduction and induction could be integrated and how they would cooperate, complement and affect each other is emerging as an important problem.

In [7] we have proposed an approach for the integration of induction and abduction in intensional ILP systems. We also suggested a way to learn a limited form of integrity constraints, namely binary constraints between a predicate and its negation. Moreover, the algorithm proposed there may introduce new abducible predicates in order to deal with exceptions to rules.

In [10] the issues of how the inductive process can be improved by exploiting abduction are discussed. Abduction can help induction not only for learning from incomplete data but also for overcoming some of the limitations of existing extensional and intensional ILP systems when learning multiple predicates and logic programs with negation. The authors propose an intensional algorithm for learning abductive logic program and suggest a way in which it can be integrated with existing learning systems in order to learn integrity constraints in their general form.

As concerns the integration of abduction and induction, a notable work is that by Dimopoulos and Kakas [5]. In this paper, the authors suggest a methodology for the integration of abduction in learning, where abduction is used first to explain the training data of a learning problem in order to generate suitable or relevant background data on which to base the inductive generalization. The main difference with our approach is that they use abduction and induction separately one after the other. Abduction plays a preparatory role for the inductive process since it abductively completes the training set starting from observations regarding examples. Our approach, instead, intertwines abduction and induction during the learning process so that unknown literals are abduced in order to cover positive examples and rule out negative ones.

In [2], the authors propose an algorithm for learning normal logic programs obtained by the substitution of abduction with induction in an abductive proof procedure, namely SLDNFA. In our framework, the two techniques are mixed in order to learn not only normal but also abductive logic programs.

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. As [1], we do not rely on any oracle, but rather on abductive proof procedure for determining the proof of an atom. Furthermore, both RUTH and our framework can treat as abducibles *some* of the program predicates. Differently from [1], we avoid clause retraction.

6 Conclusions and Future Work

In this paper, we propose a framework where abduction and induction are combined, thus solving the problems of learning in presence of incomplete knowledge. In the devised framework, the user can partially specify background and target predicates. We rely on a three-valued logic in which an atom can be *true*, *false* or *unknown*. Unspecified information are considered unknown and possibly abduced (as true or false) during the learning process in order to cover positive examples and rule out negative ones. In this way, not only we enlarge the content of the background knowledge, but also improve the learning process.

We propose an extension of FOIL, an extensional top-down ILP system, by means of an abductive proof procedure. We have shown, by means of examples, that the extended system can cope with missing knowledge.

A number of issues are subject for future work. First, we have to further investigate new heuristics for the selection of abductive literals. Unknown tuples should be given a smaller weight with respect to defined ones. Second, we will extend FOIL in order to learn also integrity constraints, as proposed, for instance, in [10].

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. F. Bergadano and D. Gunetti. *Inductive Logic Programming.* MIT press, 1996.
4. A. Brogi, E. Lamma, P. Mancarella, and P. Mello. An Abductive Framework for Extended Logic Programming. In *Proceedings 3rd Int. Workshop on Logic Programming and Non Monotonic Reasoning*, 1995.
5. Y. Dimopoulos and A. Kakas. Abduction and Learning. In *Advances in Inductive Logic Programming.* IOS Press, 1996.
6. K. Eshghi and R.A. Kowalski. Abduction compared with Negation by Failure. In *Proceedings of ICLP89*, 1989.
7. 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.
8. A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2:719–770, 1993.
9. A.C. Kakas and P. Mancarella. On the relation between Truth Maintenance and Abduction. In *Proceedings of PRICAI90*, 1990.
10. A.C. Kakas and F. Riguzzi. Abductive Concept Learning. Technical Report TR-96-15, University of Cyprus, Computer Science Department, 1996.
11. L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In *Advances in Inductive Logic Programming.* IOS Press, 1996.
12. R. Michalski, J.G. Carbonell, and T.M. Mitchell (eds). *Machine Learning - An Artificial Intelligence Approach.* Springer Verlag, 1984.
13. R. Michalski, J.G. Carbonell, and T.M. Mitchell (eds). *Machine Learning - An Artificial Intelligence Approach Vol. II.* Morgan Kaufmann, 1986.
14. D.L. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 32, 1988.
15. J. R. Quinlan and R.M. Cameron-Jones. Induction of Logic Programs: FOIL and Related Systems. *New Generation Computing*, 13:287–312, 1995.

This article was processed using the LaTeX macro package with LLNCS style