A Hybrid Extensional/Intensional System for Learning Multiple Predicate Learning and Normal Logic Programs

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] Dipartimento di Ingegneria, Università di Ferrara
Via Saragat 1, 41100 Ferrara, Italy
pmello@ing.unife.it

Abstract. We present an approach for solving some of the problems of Inductive Logic Programming systems when learning multiple predicates and normal logic programs. The approach extends the algorithm for learning abductive logic programs proposed in [7] and refined in [10] by introducing a hybrid form of coverage in which both the examples and the theory learned so far are used in the derivation of examples. We show that with hybrid coverage and abduction we are able to solve the problem of global inconsistency of intensional systems when learning multiple predicates and of non-monotonic coverage of positive example when learning normal logic programs.

1 Introduction

Most logic programs contain the definition of several predicates using negative literals in clause bodies. However, most Inductive Logic Programming (ILP) systems have been designed for learning definite clause definitions for a single predicate. Learning multiple predicates and learning logic programs with negation (*normal logic prgorams*) are two difficult tasks that create problem to most ILP systems.

If we synthesize multiple predicates programs by applying single predicate learners, we find two problems [17]. The first is that adding a clause to a partial hypothesis can make previous clauses inconsistent. The second is that a very expensive backtracking on clause addition to the theory must be performed.

When learning normal logic programs, instead, the addition of a clause to a partial hypothesis can reduce the set of positive example covered by the hypothesis, thus making impossible to use the covering approach to learning.

In order to overcome these problems, most top-down systems (e.g. ICN [14], MULT_ICN [13], FOIL [16], FOCL [15], MIS [18] with the lazy strategy) use *extensional coverage*: the coverage verification of examples is performed by using only the current clause, the background knowledge and the training set, but not previously learned clauses. In this way, clauses are learned independently from each other. We will distinguish between *extensional* and *intensional systems* depending on whether they use extensional coverage or not. However, extensional coverage introduces other problems because the learning algorithm can be unsound: the learned theory can be both inconsistent and incomplete.

We propose a learning algorithm that use abduction and a hybrid extensional-intensional coverage in order to overcome the problems of intensional system when learning multiple predicate and normal logic programs while avoiding the pitfalls of extensional systems.

The algorithm we propose is obtained by modifying that presented in [7] for learning abductive logic programs. The problem of learning abductive logic programs is emerging as a promising research direction in the field of ILP. A number of works [7, 10, 12, 11] have started to appear on the subject, and, more generally, on the relation existing between abduction and induction and how they can integrate and complement each other [5, 6, 1].

In our system, abduction plays a double role. First, it is used in order to remember relevant assumptions that are done when learning a clause and that constrain successive clauses.

Second, abduction is used in order to introduce extensionality: the training set is considered as a set of abduced literals that is taken as input by the abductive proof procedure used for deriving examples. The abduced literals are considered as additional facts that are true in the theory. By adopting a hybrid coverage, we reduce the need of backtracking on clause addition.

The paper is organized as follows: in section 2 we present the various problems of intensional, when learning multiple predicates and normal logic programs. In section 3 we discuss the problem of extensional systems. Section 4 desribes the approach for the integration of abuction and induction. Section 5 presents the abductive inductive algorithm modified in order to introduce extensionality. In section 6 we show, by means of examples, that the proposed algorithm successfully solves the above mentioned problems. In section 7 we discuss related works and in section 8 we conclude and present the directions for future work.

2 Intensional Systems

Let us first recall the definition of ILP problem [2] and the basic top-down algorithm that is shared by most intensional ILP systems.

Definition 1 ILP Problem.
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 (*background 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^-*).

With a great deal of approximation, top-down ILP systems share a common basic algorithm [2]:

```
$T := \emptyset$
**while** $E^+ \neq \emptyset$ do (Covering loop)
    Generate one clause $C$
    Remove from $E^+$ the $e^+$ covered by $C$
    Add $C$ to $T$

Generate one clause $C$ (specializing loop):
Select a predicate $p$ that must be learned
Set clause $C$ to be $p(\overline{X}) \leftarrow$ .
**while** $C$ covers some negative example **do**
    Select a literal $L$ from the language bias
    Add $L$ to the body of $C$
    Test coverage 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 $L$)
```

The algorithm above iteratively adds a clause to the current partial theory. It generates a clause by searching depth-first in the space of possible clauses. However, backtracking on clause addition is required, because otherwise the system is not garanteed to be *complete*, i.e. to find a solution if it exists. In fact, we may add to the theory a certain number of clauses and then find out that no other clause is available in the language bias for covering the remaining positive examples, while with a different choice of previous clauses we could have had a solution. This problem arises both when learning single recursive predicates, because clauses depend on each other, and when learning multiple predicates, both in the case of definite or normal logic programs. However, when learning multiple predicates the problem is more evident because the dependency relations between different clauses are more frequent and complex.

2.1 Learning Multiple Predicates

When learning multiple predicates, we have to distinguish between two types of consistency of a clause: *relative local* and *relative global consistency* of a new clause with respect to the theory learned so far (*hypothesis*). These definitions are based on the absolute definition of *local* and *global consistency* of a clause given in [17]. We will first give some terminology and define the function $covers(B, H, E)$ [17], then we will recall the definition of local and global consistency as given in [17] and finally we will give the definitions of relative local and relative global consistency.

Let the training set be $E = E^+ \cup E^-$ where E^+ is the set of positive example and E^- is the set of negative example. We assume that E contains examples for m target predicates $p_1, \ldots, p_m$ and we partition E^+ and E^- in $E^+_{p_i}$ and $E^-_{p_i}$ according to these predicates. The *hypothesis* H is a set of clauses for all the target predicates.

Definition 2 *$covers(B,H,E)$***.** Given the background theory B, the hypothesis H and the example set E, $covers(B,H,E) = \{e \in E \mid B \cup H \models e\}$

Definition 3 Global consistency. Clause c is *globally consistent* if and only if $covers(B,\{c\},E^-) = \emptyset$.

Definition 4 Local consistency. Clause c for the predicate p_i is *locally consistent* if and only if $covers(B,\{c\},E^-_{p_i}) = \emptyset$.

Definition 5 Relative global consistency. Given a consistent hypothesis H, clause c is *globally consistent with respect to H* if and only if $covers(B,H \cup \{c\},E^-) = \emptyset$.

Definition 6 Relative local consistency. Given a consistent hypothesis H, clause c for the predicate p_i is *locally consistent with respect to H* if and only if $covers(B,H \cup \{c\},E^-_{p_i}) = \emptyset$.

The basic top-down algorithm has been designed for learning single predicates: it generates a theory by iteratively adding a relatively locally consistent clause to the current partial theory. However, when learning multiple predicates, adding a relatively locally consistent clause to a consistent hypothesis can produce an inconsistent hypothesis as it is shown in the next example inspired to [17].

Example 1. We want to learn the definitions of $ancestor$ and $father$ from the knowledge base
$B = \{parent(a,b), parent(b,c), male(a), female(b)\}$
and the training set
$E^+ = \{ancestor(a,b), ancestor(b,c), ancestor(a,c), father(a,b)\}$
$E^- = \{ancestor(b,b), ancestor(b,a), ancestor(c,b), father(b,c), father(a,c)\}$
Suppose that the system has first generated the rules:
$ancestor(X,Y) \leftarrow parent(X,Y).$
$father(X,Y) \leftarrow ancestor(X,Y), male(X).$
Clearly the second rule is incorrect but the system has no mean of discovering it now, since it is locally and globally consistent with respect to the partial definition for $ancestor$.
Then the system learns the recursive rule for $ancestor$:
$ancestor(X,Y) \leftarrow parent(X,Z), ancestor(Z,Y).$
This clause is locally consistent with respect to the current hypothesis because none of the negative examples for $ancestor$ will be covered, but it is not globally consistent because the negative example $father(a,c)$ will be covered.

Therefore, in intensional systems, it is not enough to check the local consistency of a clause, but the global consistency must be checked, as it is done in the system MPL [17]. This is equivalent to test the coverage of the negative examples for all target predicates, that has a high computational cost.

2.2 Learning Normal Logic Programs

When learning normal logic programs, apart from the problem of backtracking on clause addition and global inconsistency, another problems can arise in intensional systems. Adding a clause to a partial hypothesis can reduce the coverage of that hypothesis, as it is shown in the next example. This is the dual problem of global inconsistency for definite logic programs. The covering approach of the top down algorithm is not appropriate since we can not anymore discard covered examples.

Example 2. Suppose we want to learn the definition of *member* and *intersection* from a background knowledge of definitions for $null(X)$, $head(X,Y)$, $tail(X,Y)$ and $assign(X,Y)$ and from the training set:
$E^+ = \{intersection([],[1,2],[]), intersection([2,3,4],[2,3],[2,3],$
$member(1,[1]), member(3,[2,3]), \ldots\}$
$E^- = \{intersection([3],[2,3],[2,3]), intersection([4,3,5],[4,6],[4,6]),$
$intersection([],[3,4],[2]), intersection([3],[],[4,5]),$
$member(1,[]), member(2,[1,3])\}$
Suppose the system has first generated the rules:
$member(X,Y) \leftarrow head(X,Y).$
$intersection(X,Y,Z) \leftarrow null(X), null(Z).$
$intersection(X,Y,Z) \leftarrow head(X,XH), not\ member(XH,Y), assign(Y,Z).$
The last rule is clearly incorrect but is consistent with respect to the current hypothesis and the negative examples. It covers the only positive example $intersection([2,3,4],[3,2],[3,2])$ and therefore could be generated by an intensional system.
Then the system generates the recursive clause for *member*:
$member(X,Y) \leftarrow tail(Y,YT), member(X,YT)$
that of course is locally consistent. When adding this last clause to the theory, however, the positive example $intersection([2,3,4],[3,2],[3,2])$ is no more covered by the theory.

3 Extensional Systems

Many top-down ILP systems use extensional coverage in order to solve the above mentioned problems of intensionality.

Definition 7 Extensional coverage. Given the background theory B and the example e belonging to the example set E, The clause $c = l \leftarrow l_1, l_2 \ldots l_n$ *extensionally covers* e iff l unifies with e with substitution θ and $[l_i]\theta \in \mathcal{M}(B) \cup E^+$ for $i = 1 \ldots n$.

Extensional coverage makes the evaluation of a clause independent from previous ones. Therefore we do not need anymore to backtrack on caluse addition and to search the space of possible programs, it is sufficient to iteratively search the smaller space of possible clauses.

Extensional coverage solves the problem of globally inconsistency when learning multiple predicates. In fact, by using extensional coverage, in example 1 the second rule would not be generated because all the positive examples for $ancestor$ would be used in the testing of negative examples for $father$.

Extensional coverage solves also the problem of coverage reduction when learning normal logic programs. In fact in example 2 the incorrect clause for $intersection$ would not be generated, since it would cover the negative example $intersection([3],[2,3],[2,3])$ because the positive example $member(3,[2,3])$ is used in the derivation.

However, extensional coverage poses a number of other problems. They are due to the fact that the learned theory is tested differently from the way in which it is effectively used. In particular, for definite logic programs, we can have the following cases [17]: (i) extensional consistency, intensional inconsistency; (ii) intensional completeness, extensional incompleteness; (iii) extensional completeness, intensional incompleteness. For normal logic programs we can this problems (but for dual causes) plus a new one: extensional inconsistency, intensional consistency. Let us illustrate each of these cases with an example, as it is done in [17].

Example 3 Extensional consistency, intensional inconsistency. Consider the problem of learning the concept $father$ and $male\_ancestor$ from a background knowledge containing facts about $parent$, $male$ and $female$. The training set is specified as follows: for $father$, E contains as negative examples only the facts of the form $father(a,b)$ for which $parent(a,b)$ is not in the background knowledge; for $male\_ancestor$, it contains a sufficient number of positive and negative examples. In this case, the following hypothesis is extensionally consistent but not intensionally consistent:

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

because negative examples of $male\_ancestor(a,b)$ with $female(a)$ and $parent(a,b)$ in the background will be covered.

We have the case of intensional completeness, extensional incompleteness when a hypothesis intensionally covers all the positive examples but not extensionally because some example needed for covering other examples is missing from the training set.

Example 4 Intensional completeness, extensional incompleteness. Consider the background knowledge and training set:

$B = \{parent(john, steve), parent(bill, john), parent(john, mike), parent(mike, sue)\}$
$E^+ = \{ancestor(john, steve), ancestor(bill, steve), ancestor(john, sue)\}$

The theory:

$ancestor(X,Y) \leftarrow parent(X,Y).$
$ancestor(X,Y) \leftarrow ancestor(X,Z), parent(Z,Y).$

is intensionally complete but extensionally incomplete because it does not cover the example $ancestor(john, sue)$ since the positive example $ancestor(john, mike)$ is missing.

The case of extensional completeness, intensional incompleteness occurs when we learn a program with an infinite recursive chain.

Example 5 Extensional completeness, intensional incompleteness. Given the training set:
$E^+ = \{even(0), odd(1)\}$
and the background predicate $succ(X, Y)$ that expresses that Y is the successor of X, the program:
$even(X) \leftarrow succ(X, Y), odd(Y).$
$odd(X) \leftarrow succ(Y, X), even(Y).$
is extensionally complete but intensionally incomplete, because the intenesional derivation of $even(0)$ would leed to a loop.

When learning normal logic programs, extensional systems suffer also from the problem of extensional inconsistency, intensional consistency.

Example 6 Extensional inconsistency, intensional consistency. Suppose you are given the training set
$E^+ = \{intersection([3], [2, 4], []), \ldots\}$
$E^- = \{intersection([4, 3], [2, 4], []), \ldots\}$
where E^+ does not contain the example $member(4, [2, 4])$. The program
$member(X, Y) \leftarrow head(X, Y).$
$member(X, Y) \leftarrow tail(Y, YT), member(X, YT)$
$intersection(X, Y, Z) \leftarrow null(X), null(Z).$
$intersection(X, Y, Z) \leftarrow head(X, XH), tail(X, XT), member(XH, Y), intersection(XT, Y, W), cons(XH, W, Z$
$intersection(X, Y, Z) \leftarrow head(X, XH), tail(X, XT), not\ member(XH, Y), intersection(XT, Y, Z).$
is intensionally consistent but extensionally inconsistent because the negative example $intersection([4, 3], [2, 4], [])$ is extensionally covered, since $member(4, [2, 4])$ is not in E^+.

Our system does not suffer from the problem of extensional systems apart from the problem of extensional completeness, intensional incompleteness. A solution to this problem has been proposed in [13] with the system MULT_ICN. That solution can be easily integrated in our system and is subject for future work.

4 Integrating Abductive and Inductive Logic Programming

In this section, we recall the approach for the integration of abduction and induction that was proposed in [7, 12]. First, we summarize the main concepts of Abductive Logic Programming (ALP) and then we show how the learning problem of ILP must be modified in order to integrate abduction.

4.1 Abductive Logic Programming

We first give the definition of Abductive Logic Program.

Definition 8 Abductive Logic Program. An *abductive logic program* is a triple $\langle P, A, IC \rangle$ where

- P is a normal logic program,
- A is a set of *abducible predicates*,
- IC is a set of integrity constraints in the form of denials, i.e.:
  $\leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_{m+n}$.

Abducible predicates are used to model incompleteness: these are predicates for which a definition may be missing or for which the definition may be incomplete. This are the predicates about which we can make assumptions is order to explain the current goal. More formally, given an abductive program $AT = \langle P, A, IC \rangle$ and a formula G, the goal of abduction is to find a (possibly minimal) set of ground atoms Δ (*abductive explanation*) for predicates in 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. $P \cup \Delta \models IC$. We say that AT *abductively entails* e ($AT \models_A e$) when there exists an abductive explanation for e from AT. We adopt the three-valued semantics for ALP defined in [4] in which an atom can be *true, false* or *unknown*. In particular, the semantics $\mathcal{M}_{AT}$ of a program AT is defined in terms of three sets:

- $\mathcal{M}^+_{AT}$, the set of ground atoms *true* for AT,
- $\mathcal{M}^-_{AT}$, the set of ground atoms *false* for AT,
- $\mathcal{M}^u_{AT} = \overline{\mathcal{M}^+_{AT} \cup \mathcal{M}^-_{AT}}$, the set of ground atoms *unknown* for AT.

The semantics $\mathcal{M}_{AT}$ is the set of ground *literals* true for AT and is given by $\mathcal{M}_{AT} = \mathcal{M}^+_{AT} \cup not\_\mathcal{M}^-_{AT}$ where $not\_\mathcal{M}^-_{AT} = \{not\_a | a \in \mathcal{M}^-_{AT}\}$. In this way we can model domains where the knowledge is not complete.

Negation as Failure is replaced, in ALP, by Negation by Default and is obtained, through abduction, in this way: for each predicate symbol p, a new predicate symbol not_p is added to the set 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. We define the *opposite* $\bar{l}$ of a literal l as

$$\bar{l} = \begin{cases} not\_p(\mathbf{X}) & \text{if } l = p(\mathbf{X}) \\ p(\mathbf{X}) & \text{if } l = not\_p(\mathbf{X}) \end{cases}$$

Operationally, we rely on the proof procedure defined by Kakas and Mancarella [9]. This procedure starts from a goal and a set of abduced literals Δ_{in} and results in a set of consistent assumptions Δ_{out} (*abduced literals*) such that $\Delta_{out} \subseteq \Delta_{in}$ and Δ_{out} together with the program allow to derive the goal. We write

$$AT \vdash^{\Delta_{out}}_{\Delta_{in}} G$$

The correctness of this proof procedure with respect to the abductive semantics defined in [4] is established by soundness and completeness theorems in [4]. 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.

The proof procedure consists of two parts: an abductive and a consistency phase (see Appendix for the detailed algorithm). Basically, the abductive phase differs from a standard Prolog derivation when the literal to be reduced is abducible. First checks to see if the abducible literal has already been assumed (i.e., it is in the Δ set), in this case the literal is reduced or the derivation fails if the opposite of the literal is in the Δ. If it has not yet been abduced, the procedure tries to abduce it and checks that it is consistent with the integrity constraints and with the current Δ by adding the it to Δ and by starting a consistency derivation.

The first step of the consistency derivation consist in finding all the integrity constraints (denials for simplicity) in which the literal is contained. In order to abduce the literal, all these constraints must be satisfied. A denial fails only if all its conjuncts are true, therefore at least one conjunct must be false. Since one wants to assume the literal true, the algorithm removes it from the constraints and checks that all the remaining goals fail. The goals are reduced literal by literal: if a literal is abducible, first it is checked if the literal itself is already in Δ (in that case the literal is dropped) or if its opposite is already in Δ (in that case the constraint is satisfied and is no more considered). If the literal is not in Δ, an abductive derivation for its opposite is started, so that if it succeeds the constraint is satisfied.

In order to illustrate the behaviour of the abductive proof procedure, let us consider a classic example inspired to [?].

Example 7. Suppose to have the following abductive theory:
$P = \{shoes\_are\_wet \leftarrow grass\_is\_wet.$
$grass\_is\_wet \leftarrow sprinkler\_was\_on.$
$grass\_is\_wet \leftarrow rained\_last\_night.$
$electrical\_black\_out.\}$
$A = \{rained\_last\_night, sprinkler\_was\_on.\}$
$IC = \{\leftarrow electrical\_black\_out, sprinkler\_was\_on.\}$
The observation $shoes\_are\_wet$ can be only explained by the set of assumptions $\{rained\_last\_night\}$. Let us see in some detail how the abductive proof procedure works. An abductive phase is started for the goal $\leftarrow shoes\_are\_wet$ with $\Delta = \emptyset$. Then the goal is unfolded with the first rule giving the resolvent:
$\leftarrow sprinkler\_was\_on.$
Since there are no rules for predicate $sprinkler\_was\_on$ and this predicate is abducible, a consistency derivation is started for it with $\Delta = \{sprinkler\_was\_on\}$. First all the constraints containing the literal are considered (respectively, $\leftarrow sprinkler\_was\_on, not\_sprinkler\_was\_on$[3] and $\leftarrow electrical\_black\_out, sprinkler\_was\_on.$) and then the goal is unfolded with the constraint giving $\leftarrow not\_sprinkler\_was\_on$

[3] This constraint was added when transforming Negation as Failure literals in Negation as Default literals.

and $\leftarrow electrical\_black\_out$. Now, both these goals must fail for the consistency to succeed. The first goal clearly fails since the opposite of the literal is in Δ, but the second goal succeeds. Therefore the consistency fails and in bactracking the initial goal $shoes\_are\_wet$ is unfolded with the second rule, giving $\leftarrow rained\_last\_night$. This time the consistency derivation for $rained\_last\_night$ succeeds and therefore the outer abductive derivation succeeds as well with $\Delta = \{rained\_last\_night\}$.

Let us now see how default negation goals are treated. The goal:

$\leftarrow not\_grass\_is\_wet$

succeeds with the abduction of $\Delta = \{not\_rained\_last\_night, not\_sprinkler\_was\_on\}$. In fact, the abductive derivation for $\leftarrow not\_grass\_is\_wet$ immediately starts a consistency for $not\_grass\_is\_wet$ (since all default literals are abducible) with $\Delta = \{not\_grass\_is\_wet\}$. Unfolding it with the only relevant constraint gives $\leftarrow grass\_is\_wet$ that is solved in all possible ways giving the goals:

$\leftarrow sprinkler\_was\_on.$

$\leftarrow rained\_last\_night.$

that must all fail. Since they both contain abducible literals, an abductive derivation is started first for $\leftarrow not\_sprinkler\_was\_on.$ and then for $\leftarrow not\_rained\_last\_night.$. Both of them succeed abducing:

$\Delta = \{not\_rained\_last\_night, not\_sprinkler\_was\_on\}$

4.2 New Learning Problem

We consider a new definition of the ILP learning problem similar to Abductive Concept Learning (ACL) [6]. In this extended learning problem both the background and target theory are abductive theories and the notion of deductive entailment is replaced by abductive entailment.

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 $AT' = \langle T', A, IC\rangle \in \mathcal{P}$ with $T' \supseteq T$, such that

let $E = E^+ \cup \{not\_e \mid e \in E^-\}$

$\forall e \in E, \quad AT' \vdash_{\emptyset}^{\Delta_e} e$

$\bigcup_{\forall e \in E} \Delta_e \cup T' \models IC$ (consistency of the assumptions w.r.t. IC)

When $AT \vdash_{\emptyset}^{\Delta_e} e$ we say that AT *abductively covers* e under hypotheses Δ_e.

The abductive program that is learned can contain new rules (possibly with abducibles in the body) but not new abducible predicates and new integrity constraints.

In order to introduce extensional coverage in this framework, we require all target predicates to be abducible and we change the condition that the learned program must satisfy in this way:

$$\forall e \in E,\ AT' \vdash^{\Delta_e}_{E \setminus \{e\}} e$$
$$\bigcup_{\forall e \in E} \Delta_e \cup T' \models IC$$

Differently from def. 7 for extensional coverage, here also negative examples are used because of the three-valued semantics of abduction [4]. The literal l is proved true if $l \in E$ and is proved false if $\overline{l} \in E$.

5 The hybrid algorithm

In this section, we present an intensional algorithm that is able to learn abductive logic programs [11] and we show how it can be extended, by exploiting abduction, to incorporate extensional coverage. The algorithm is reported in figures 1, 2, 3 and is obtained from the basic top-down ILP algorithm [2] (see also section [11]), by substituting the usual notion of coverage of examples with the notion of abductive coverage.

```
procedure LAP(
    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 (covering loop)
    GenerateRule(input: $AT, H, E^+, E^-, \Delta$; output: $Rule, E^+_{Rule}, \Delta_{Rule}$)
    Move to $E^+$ all the positive literals of target predicates in $\Delta_{Rule}$
    Move to $E^-$ all the atoms corresponding to
        negative literals of target predicates in $\Delta_{Rule}$
    $E^+ := E^+ - E^+_{Rule}$
    $H := H \cup \{Rule\}$
    $\Delta := \Delta \cup \Delta_{Rule}$
endwhile
output $H$
```

Fig. 1. The covering loop

The basic top-down algorithm is extended in the following respects in order to learn abductive logic programs. First, in order to test the coverage of the generated rule, (procedure TestCoverage in figure 3) an abductive derivation is started for each positive example and the default negation (not_e^-) of each negative (e^-). Each derivation starts from the set of literals abduced in the derivations of the previously covered examples. In this way, we ensure that the assumptions made during the derivation of the current example (positive or negative) are consistent with the assumptions previously raised for deriving other examples.

```
procedure GenerateRule(
    inputs : $AT, E^+, E^-, H, \Delta$
    outputs : $Rule$ : rule,
        $E^+_{Rule}$ : positive examples covered by $Rule$,
        $\Delta_{Rule}$ : abduced literals )

Select a predicate $p$ to be learned
Let $Rule = p(X) \leftarrow true$.
repeat (specialization loop)
    select a literal $L$ from the language bias
    add $L$ to the body of $Rule$
    TestCoverage(input: $Rule, AT, H, E^+, E^-, \Delta$,
        output: $E^+_{Rule}, E^-_{Rule}, \Delta_{Rule}$)
    if $E^+_{Rule} = \emptyset$
        backtrack to a different choice for $L$
until $E^-_{Rule} = \emptyset$
output $Rule, E^+_{Rule}, \Delta_{Rule}$
```

Fig. 2. The specialization loop

Second, after the generation of each clause, the abduced literals of target predicates are added to the training set, so that they become new training examples (figure 1).

In order to introduce extensional coverage in the algorithm, each abductive derivation of an example does not starts only with the set of literals already abduced but also with the training set itself (see also figure 3). In particular, the input abducibles are augmented with all the positive examples and the default negation of each negative. In order to avoid the trivial derivation of e^+ based on e^+ itself, e^+ is taken out from the input abducibles. The same is done for not_e^- when trying to derive not_e^-. The modifications to the procedure TestCoverage are shown in figure 3 as framed formulae.

We now show an example of the behaviour of the algorithm in the case of learning the predicate member. Let the background knowledge and training set be:

$B = \{components([H|T], H, T) \leftarrow\}$

$E^+ = \{member(2, [2]), member(2, [1, 2, 3]), member(3, [1, 2, 3])\}$

$E^- = \{member(2, []), member(2, [3]), member(1, [2, 3])\}$

Suppose the system first generates the clause

$member(A, B) \leftarrow components(B, C, D), member(A, D)$

Then the clause is tested. The abductive derivation $\leftarrow member(2, [2])$ fails because $member(2, [])$ can not be derived nor abduced, since it is a negative example. In the abductive derivation of $member(2, [1, 2, 3])$, first the system unfolds two times the clause and tries to abduce $member(2, [3])$. Since it is a negative example, the derivation fails and, in backtracking, it succeeds with the abduction of $member(2, [2, 3])$. Finally, the positive example $member(3, [1, 2, 3])$ is covered with the abduction of $member(3, [3])$. Then negative examples are

```
procedure TestCoverage(
    inputs : $Rule$ : rule, $AT = \langle T, A, IC \rangle$ : 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 $Rule$
        $\Delta_{Rule}$ : new set of abduced literals

$E^+_{Rule} = E^-_{Rule} = \emptyset$; $\Delta_{in} = \Delta$; $\boxed{E = E^+ \cup \{not\_e \mid e \in E^-\}}$
for each $e^+ \in E^+$ do
    if AbdDer($e^+, \langle T \cup H \cup \{Rule\}, A, IC \rangle, \Delta_{in} \boxed{\cup E \setminus \{e^+\}}, \Delta_{out}$)
        succeeds then add $e^+$ to $E^+_{Rule}$; $\Delta_{in} = \Delta_{out}$
for each $e^- \in E^-$ do
    if AbdDer($not\_e^-, \langle T \cup H \cup \{Rule\}, A, IC \rangle, \Delta_{in} \boxed{\cup E \setminus \{not\_e^-\}}, \Delta_{out}$)
        succeeds then $\Delta_{in} = \Delta_{out}$
    else add $e^-$ to $E^-_{Rule}$
$\Delta_{Rule} = \Delta_{out} \setminus \Delta$
output $E^+_{Rule}, E^-_{Rule}, \Delta_{Rule}$
```

Fig. 3. Hybrid coverage testing

tested: $\leftarrow not\_member(2, [])$, $\leftarrow not\_member(2, [3])$ and $\leftarrow not\_member(1, [2, 3])$ all succeeds. In the last case, $not\_member(1, [3])$ is abduced. Therefore the rule is consistent and is added to the hypothesis. Covered positive examples are removed and assumptions about target predicates are added to the training set, that becomes:

$E^+ = \{member(2, [2]), member(2, [2, 3]), member(3, [3])\}$

$E^- = \{member(2, []), member(2, [3]), member(1, [2, 3]), member(1, [3])\}$

Then the system generates the clause

$member(A, B) \leftarrow components(B, A, D)$

that covers all the remaining positive examples and the negation of the negative ones without abducing anything. The clause is added to the hypothesis and the algorithm terminates.

6 Properties of the Algorithm

In this section, we illustrate by means of examples the way in which the algorithm satisfies the above mentioned properties. We will first show that the hybrid system does not suffer from the problem of extensional consistency, intensional inconsistency and intensional completeness, extensional incompleteness of extensional system. Then we will demonstrate that the system does not generate globally inconsistent hypothesis and does reduce the coverage of the current hypothesis.

Let us consider first definite logic programs. For them, we do not have the problem of extensional consistency, intensional inconsistency, as shown in example 3, because each negative example is tested also against the current hypothesis.

We do not have neither the problem of intensional completeness, extensional incompleteness, as shown in example 5. If a positive example is not covered because a needed literal for a target predicate is missing in training set, the intensional definition for the target predicate will be used instead.

As regards normal logic programs, with similar reasoning it is possible to show that a hybrid system do not suffer from the same problems, apart from the problem of extensional completeness, intensional incompleteness due to loops through recursion.

6.1 Learning Multiple Predicates

Let us now turn to the problem of global inconsistency. There are mainly two cases in which we can incur in this problem. The first is the one shown in example 1, in which the rule at fault is in the hypothesis before the addition of the newly generated rule, while the second case is the one in which the rule at fault is the new one.

The first case is dealt with by using a hybrid system. In example 1 the rule about *father* would not be generated using a hybrid system because it would have used as well the examples to complete the definition for *ancestor*, thus discovering that the rule for *father* would cover some negative example. Therefore, with a hybrid system the situation in example 1 would never occur.

However, another situation may occour, in which we have a "correct" partial hypothesis (in the sense that it is part of the final solution) that is made globally inconsistent by a newly added rule. In this case, we prevent the system from learning a globally inconsistent hypothesis by using abduction. Let us illustrate the point by considering a more general case. Suppose the system has learned the following clauses in the order in which they are listed:

$q(X) \leftarrow Body_1(X).$
$p(X) \leftarrow Body_2(X), q(X).$
$q(X) \leftarrow Body_3(X).$

Suppose that the second clause is consistent since it rules out all the negative examples e_p^- for p because for them $q(e_p^-)$ is always false, either because it is in E^- or because it is not derivable. When we add the third clause, this can cover as well some of the atoms $q(e_p^-)$, because they may not be in E_q^-.

We avoid this problem in the following way. When we test the clause:

$p(X) \leftarrow Body_2(X), q(X).$

against a negative example $p(e_p^-)$, we start an abductive derivation for $not\_p(e_p^-)$. If $q(e_p^-)$ is not derivable, the derivation succeeds with the abduction of $not\_q(e_p^-)$ that is added in the Δ set. After the clause is added to the theory, all the abduced literals regarding target predicates are moved to the training set: thus $q(e_p^-)$ will be added to E^-. In this way the system will not be able to generate a clause $q(X) \leftarrow Body_3(X)$. that covers $q(e_p^-)$ and makes the previous clauses globally inconsistent.

Therefore, in our system both abduction and hybrid coverage play an important role in the solution of the problem of global inconsistency. Thanks to hybrid coverage we avoid the generation of "incorrect" clause based on a partial definition of a subpredicate, that should later be retracted, thus avoiding the use of an expensive backtracking. Besides being used to implement hybrid coverage, abduction prevents also from learning apparently "correct" clauses that would otherwise be very difficult to identify, later on, as the source of inconsistency.

6.2 Learning Normal Logic Programs

Let us now turn to the problem of coverage reduction. This problem as well can appear in two cases: the first is the one shown in example 2, in which the rule at fault is already in the hypothesis while the second is the one in which the rule at fault is the new one.

The first case is dealt with by using a hybrid system. In example 2 the first rule about *intersection* would not be generated using a hybrid system because it would have used as well the examples to complete the definition for it.

As regards the second case, consider an example in which the system has learned the following clauses in the order in which they are listed:

$q(X) \leftarrow Body_1(X).$

$p(X) \leftarrow Body_2(X), not\ q(X).$

$q(X) \leftarrow Body_3(X).$

After the addition of the third clause, some of the positive examples covered by the second may not be covered anymore. A similar problem can arise also in single predicate learning, in the case of negative recursive clauses:

$p(X) \leftarrow Body_1(X).$

$p(X) \leftarrow Body_2(X, Y), not\ p(Y).$

$p(X) \leftarrow Body_3(X).$

The problem arise because the set of positive examples is gradually reduced and covered positive examples are no longer tested. This is the dual problem of the one seen before for definite programs (see section 6.1). The learning process is again non-monotonic but this time with respect to coverage instead of consistency. Therefore, we can not take out the positive examples from E^+ when they are covered by a clause, but after the addition of each clause all the previously covered e^+ must be checked again.

When testing the positive example $p(e_p^+)$, the system records assumptions about negative literals *not* $p(e_p^+)$ being true (or $p(e_p^+)$ being false) by storing them in the Δ set. These assumption will then be moved to the training set so that clauses generated afterwards will not cover the example $p(e_p^+)$.

The system avoids the problem of global inconsistency also for normal logic program because (positive or negative) assumptions made during the testing of positive examples are recorded and added to the training set.

Per la negazione, provare ad imparare hamilton (vedi tracynot).

7 Related Work

This work was inspired by [10] where the algorithm for learning abductive logic programs was introduced and its main properties studied. We improved on that work by adding the hybrid coverage to the system, that has the important property of avoiding backtracking and thus rendering practically feasible the learning of multiple predicates and normal logic programs.

On the problem of learning multiple predicates a notable work is [17] where the authors thoroughly analyze the problem and the solutions proposed both by intensional and extensional systems. Moreover, they present the clear and extensive analisys of the various problems of extensional systems that we have recalled in section 3. In order to overcome the problem of global incompleteness, they propose the system MPL that takes a different approach with respect to ours. Being an intensional system, it can not avoid the case described in example 1 as we do. Also in the other case of global inconsistency, it solves the problem by retracting one of the globally inconsistent clauses previously learned. However, if the clause at fault is the last one, MPL would retract the wrong clause and would not be able to learn the desired theory because retracted clauses cannot be added again to the hypothesis.

Our system still suffers from the problem of extensional completeness, intensional incompleteness. This problem has been deeply studied, both for definite and normal logic program, in [14]. The authors propose the system ICN in which they solve the problem by keeping explicit track of the recursive dependency among clauses. An interesting direction for future work would be to incorporate their solution into our system.

Hybrid coverage is used as well in the system FOIL-I [8]. However, FOIL-I's authors especially concentrate on learning recursive predicates from a sparse training set and they do not investigate the properties of such a system with respect to multiple predicate learning and learning normal logic programs.

The approach we adopted to avoid the problem of coverage restriction when learning normal logic program is similar to the one followed in TRACY^{not} [3]. When learning definite logic programs, TRACY finds a hypothesis by considering the trace of the derivation of the first positive example against the set of all possible clauses. Then the clauses involved in the derivation of the example are tested against all the other examples, positive and negative. They will be part of the solution if they do not cover any negative example. This approach is based on the fact that definite logic programs are monotonic and therefore having all the clauses together does not constitute a problem because the irrelevant clauses will not interfere with the relevant ones. When learning normal logic programs, instead, irrelevant clauses may constitute a problem to relevant ones, because Negation As Failure (NAF) literals may fail because of an irrelevant clause. Therefore it is not possible to consider a trace as a possible solution. In order to overcome this problem, TRACY^{not} modifies the proof procedure of the example so that each time a NAF literal is encountered, it is removed from the resolvent and its positive version is added to the training set: it is added to E^- if the example under test was positive or to E^+ if the example was negative. In the case in which one of the negative examples is covered by the trace, TRACY^{not} performs a backtracking on the trace for the original positive examples. This is very similar to our approach, the only difference is the technique that is used for implementing it: by using abduction, we use a general technique that allows us to learn also multiple predicate and learning from incomplete information.

Since we gradually add negative examples, our approach may seem similar to the one adopted in incremental systems such as MIS [18]. However, while in incremental systems a consistency check must be done after the addition of each e^- to the training set, we do not have to do this because we add an e^- only after having tested that it is not covered by any clause.

8 Conclusions and Future Work

We have shown how abduction can be used in order to introduce extensionality in intensional systems. In particular, we have taken the intensional system for learning abductive logic programs proposed in [10] and we have extended it in order to include extensional coverage. In this way, we get a hybrid system that overcomes two problems of intensional systems: the problem of global inconsistency when learning multiple predicates and the problem of coverage reduction when learning normal logic programs. Moreover, we do not incur in the problems of completeness and consistency of extensional systems, apart from the one of extensional completeness, intensional incompleteness. This problem is relevant only when learning (mutually) recursive predicates, that was not our main aim. Subject for future work will be to integrate into our systems the techniques proposed in [14] for learning recursive predicates.

Acknowledgment

The authors would like to thank the anonymous referees and the participant to the workshop for their useful comments and suggestions. Fabrizio Riguzzi would like to thank Tony Kakas for having invited him to spend a period at the University of Cyprus and for the many helpful discussions has together.

References

1. H. Adé and M. Denecker. AILP: Abductive inductive logic programming. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.
2. F. Bergadano and D. Gunetti. *Inductive Logic Programming.* MIT press, 1996.
3. F. Bergadano and D. Gunetti. Learning Logic Programs with Negation as Failure. In *Advances in Inductive Logic Programming.* IOS Press, 1996.
4. A. Brogi, E. Lamma, P. Mancarella, and P. Mello. A unifying view for logic programming with non-monotonic reasoning. To appear on the Journal of Theoretical Computer Science.

5. M. Denecker, L. De Raedt, P. Flach, and A. Kakas, editors. *Proceedings of ECAI96 Workshop on Abductive and Inductive Reasoning.* Catholic University of Leuven, 1996.
6. Y. Dimopoulos and A. Kakas. Abduction and learning. In *Advances in Inductive Logic Programming.* IOS Press, 1996.
7. F. Esposito, E. Lamma, D. Malerba, P. Mello, M. Milano, F. Riguzzi, and G. Semeraro. Learning abductive logic programs. In Denecker et al. [5].
8. 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.
9. A.C. Kakas, R.A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. et al. Gabbay, editor, *Handbook of Logic in AI and Logic Programming.* 1997. to appear.
10. A.C. Kakas and P. Mancarella. On the relation between truth maintenance and abduction. In *Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence*, 1990.
11. A.C. Kakas and F. Riguzzi. Learning with abduction. Technical Report TR-96-15, University of Cyprus, Computer Science Department, 1996.
12. A.C. Kakas and F. Riguzzi. Learning with abduction. In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, 1997.
13. E. Lamma, P. Mello, M. Milano, and F. Riguzzi. Integrating Induction and Abduction in Logic Programming. In P. P. Wang, editor, *Prooceedings of the Third Joint Conference on Information Sciences*, volume 2, pages 203–206, 1997.
14. L. Martin and C. Vrain. MULT_ICN: An empirical multiple predicate learner. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 129–144. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
15. L. Martin and C. Vrain. A three-valued framework for the induction of general program. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 109–128. Department of Computer Science, Katholieke Universiteit Leuven, 1995.
16. M.J. Pazzani and D. Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9(1):57–94, 1992.
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, N. Lavrač, and S. Džeroski. Multiple predicate learning. In *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, 1993.
19. E. Shapiro. *Algorithmic Program Debugging.* MIT Press, 1983.

Appendix

In the following we recall the abductive and consistency derivation used by our algorithm, which are taken from [9].

Abductive derivation

An abductive derivation from $(G_1\ \Delta_1)$ to $(G_n\ \Delta_n)$ in $\langle P, Ab, IC\rangle$ via a selection rule R is a sequence

$$(G_1\ \Delta_1), (G_2\ \Delta_2), \ldots, (G_n\ \Delta_n)$$

such that each G_i has the form $\leftarrow L_1, \dots, L_k$, $R(G_i) = L_j$ and $(G_{i+1}\ \Delta_{i+1})$ is obtained according to one of the following rules:

(1) If it exist a resolvent C of some clause in P with G_i on the selected literal L_j, then $G_{i+1} = C$ and $\Delta_{i+1} = \Delta_i$;
(2) If L_j is abducible or default and $L_j \in \Delta_i$ then $G_{i+1} = \leftarrow L_1, \dots, L_{j-1}, L_{j+1}, \dots, L_k$ and $\Delta_{i+1} = \Delta_i$;
(3) If L_j is abducible or default, $L_j \notin \Delta_i$ and $\overline{L_j} \notin \Delta_i$ and there exists a *consistency derivation*from $(\{L_j\}\ \Delta_i \cup \{L_j\})$ to $(\{\}\ \Delta')$ then $G_{i+1} = \leftarrow L_1, \dots, L_{j-1}, L_{j+1}, \dots, L_k$ and $\Delta_{i+1} = \Delta'$.

Steps (1) is SLD-resolution. Step (2) consider the case in which the literal has already been abduced. In step (3) a new abductive or default hypotheses is required and it is added to the current set of hypotheses provided it is consistent.

Consistency derivation
A consistency derivation for an abducible or default literal α from $(\alpha,\ \Delta_1)$ to $(F_n\ \Delta_n)$ in $\langle P, Ab, IC\rangle$ is a sequence

$$(\alpha\ \Delta_1), (F_1\ \Delta_1), (F_2\ \Delta_2), \dots, (F_n\ \Delta_n)$$

where :

(i) F_1 is the union of all goals of the form $\leftarrow L_1, \dots, L_n$ obtained by resolving the abducible or default α with the denials in IC with no such goal been empty, $\leftarrow$;
(ii) for each $i > 1$, F_i has the form $\{\leftarrow L_1, \dots, L_k\} \cup F'_i$ and for some $j = 1, \dots, k$ $(F_{i+1}\ \Delta_{i+1})$ is obtained according to one of the following rules:
  (C1) If the set C' of all resolvents of clauses in P with $\leftarrow L_1, \dots, L_k$ on the literal L_j is not empty and $\leftarrow \notin C'$, then $F_{i+1} = C' \cup F'_i$ and $\Delta_{i+1} = \Delta_i$;
  (C2) If L_j is abducible or default, $L_j \in \Delta_i$ and $k > 1$, then $F_{i+1} = \{\leftarrow L_1, \dots, L_{j-1}, L_{j+1}, \dots, L_k\} \cup F'_i$ and $\Delta_{i+1} = \Delta_i$;
  (C3) If L_j is abducible or default, $\overline{L_j} \in \Delta_i$ then $F_{i+1} = F'_i$ and $\Delta_{i+1} = \Delta_i$;
  (C4) If L_j is abducible or default, $L_j \notin \Delta_i$ and $\overline{L_j} \notin \Delta_i$, and there exists an *abductive derivation* from $(\leftarrow \overline{L_j}\ \Delta_i)$ to $(\leftarrow\ \ \Delta')$ then $F_{i+1} = F'_i$ and $\Delta_{i+1} = \Delta'$.

In case (C1) the current branch splits into as many branches as the number of resolvents of $\leftarrow L_1, \dots, L_k$ with the clauses in P on L_j. If the empty clause is one of such resolvents the whole consistency check fails. In case (C2) the goal under consideration is made simpler if literal L_j belongs to the current set of hypotheses Δ_i. If $k = 1$ the consistency derivation fails. In case (C3) the current branch is already consistent under the assumptions in Δ_i, and this branch is dropped from the consistency checking In case (C4) the current branch of the consistency search space can be dropped provided $\leftarrow \overline{L_j}$ is abductively provable.

Given a query L (atomic, for the sake of simplicity), the procedure succeeds, and returns the set of abducibles Δ if there exists an abductive derivation from $(\leftarrow L\ \{\})$ to $(\leftarrow\ \ \Delta)$. With abuse of terminology, in this case, we also say that the abductive derivation succeeds.

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