1 COOPERATION OF ABDUCTION AND INDUCTION IN LOGIC PROGRAMMING

Evelina Lamma, Paola Mello, Fabrizio Riguzzi, Floriana Esposito, Stefano Ferilli, and Giovanni Semeraro

1.1 INTRODUCTION

This paper proposes an approach for the cooperation of abduction and induction in the context of Logic Programming. We do not take a stance on the debate on the nature of abduction and induction (see Flach and Kakas, this volume), rather we assume the definitions that are given in Abductive Logic Programming (ALP) and Inductive Logic Programming (ILP).

We present an algorithm where abduction helps induction by generating atomic hypotheses that can be used as new training examples or for completing an incomplete background knowledge. Induction helps abduction by generalizing abductive explanations.

A number of approaches for the cooperation of abduction and induction are presented in this volume (e.g., by Abe, Sakama, Inoue and Haneda, Mooney). Even if these approaches have been developed independently, they show remarkable similarities, leading one to think that there is a "natural way" for the integration of the two inference processes, as it has been pointed out in the introductory chapter by Flach and Kakas.

The algorithm solves a new learning problem where background and target theory are abductive theories, and abductive derivability is used as the example coverage relation. The algorithm is an extension of a basic top-down algorithm adopted in ILP (Bergadano and Gunetti, 1996) where the proof procedure

defined in (Kakas and Mancarella, 1990) for abductive logic programs is used for testing the coverage of examples in substitution of the deductive proof procedure of Logic Programming.

The algorithm has been implemented in a system called LAP (Lamma et al., 1997) by using Sicstus Prolog 3#5. The code of the system and some of the examples shown in the paper are available at

`http://www-lia.deis.unibo.it/Software/LAP/`

We also discuss how to learn abductive theories: we show that, in case of complete knowledge, the rule part of an abductive theory can be also learned without abduction. Abduction is not essential to this task, but it is essential in case of absence of information, i.e. when the background theory is abductive.

The paper is organized as follows: in section 1.2 we recall the main concepts of Abductive Logic Programming, Inductive Logic Programming, and the definition of the abductive learning framework. Section 1.3 presents the learning algorithm. In section 1.4 we apply the algorithm to the problem of learning from incomplete knowledge, learning theories for abductive diagnosis and learning exceptions to rules. Our approach to the integration of abduction and induction is discussed in detail and is compared with works by other authors in section 1.5. Section 1.6 concludes and presents directions for future work.

1.2 ABDUCTIVE AND INDUCTIVE LOGIC PROGRAMMING

In this section we recall the definitions of abduction and induction in Logic Programming given by Flach and Kakas in the introductory chapter and we add the satisfaction of integrity constraint to abduction and the avoidance of negative examples to induction.

1.2.1 Abductive Logic Programming

In a Logic Programming setting, an *abductive theory* (Kakas et al., 1997) 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.:
  $\leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_{m+n}$.

Given an abductive theory $T = \langle P, A, IC \rangle$ and a formula G, an abductive explanation Δ for G is a set of ground atoms of predicates in A such that $P \cup \Delta \models G$ (Δ explains G) and $P \cup \Delta \models IC$ (Δ is consistent). When there exists an abductive explanation for G in T, we say that T *abductively entails* G and we write $T \models_A G$.

Negation As Failure (Clark, 1978) is replaced, in ALP, by Negation by Default (Eshghi and Kowalski, 1989) and is obtained by transforming the program into its *positive version*: for each predicate symbol $p/arity$ in the

program, a new predicate symbol $not\_p/arity$ is added to the set A and the integrity constraint $\leftarrow p(\vec{X}), not\_p(\vec{X})$[1] is added to IC. Then, each negative literal $not\ p(\vec{t})$ in the program is replaced by a literal $not\_p(\vec{t})$. Atoms of the form $not\_p(\vec{t})$ are called *default atoms*. For simplicity, in the following we will write abductive theories with Negation by Default and we will implicitly assume the transformation.

We define the *complement* $\overline{l}$ of a literal l as

$$\overline{l} = \begin{cases} not\_p(\vec{x}) & \text{if } l = p(\vec{x}) \\ p(\vec{x}) & \text{if } l = not\_p(\vec{x}) \end{cases}$$

In (Kakas and Mancarella, 1990) a proof procedure for the positive version of abductive logic programs has been defined. This procedure (reported in the Appendix) starts from a goal G and a set of initial assumptions Δ_i and results in a set of consistent hypotheses (abduced literals) Δ_o such that $\Delta_o \supseteq \Delta_i$ and Δ_o is an abductive explanation of G. The proof procedure employs the notions of *abductive* and *consistency derivations*. Intuitively, an abductive derivation is the standard Logic Programming 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 is proved that any integrity constraint containing δ is satisfied. To this purpose, a consistency derivation for δ is started. Every integrity constraint containing δ is considered and δ is removed from it. The constraints are satisfied if we prove that the resulting goals fail. In the consistency derivation, when an abducible is encountered, an abductive derivation for its complement is started in order to prove its falsity, so that the constraint is satisfied.

When the procedure succeeds for the goal G and the initial set of assumptions Δ_i producing as output the set of assumptions Δ_o, we say that T *abductively derives* G or that G is *abductively derived* from T and we write $T \vdash_{\Delta_i}^{\Delta_o} G$. Negative atoms of the form not_a in the explanation have to be interpreted as "a must be false in the theory", "a cannot be assumed" or "a must be absent from any model of the theory".

In (Brogi et al., 1997) it has been proved that the proof procedure is *sound* and *weakly complete* with respect to an abductive model semantics under a number of restrictions:

- the abductive logic program must be ground,
- abducibles must not have a definition in the program,
- integrity constraints are denials with at least one abducible in each constraint.

The requirement that the program is ground is not restrictive in the case in which there are no function symbols in the program and therefore the Herbrand universe is finite. In this case, in fact, we can obtain a finite ground

[1] In the following, $\vec{X}$ represents a tuple of variables and $\vec{t}$ a tuple of terms.

program from a non-ground one by grounding in all possible ways the rules and constraints in the program.

The soundness and weak completeness of the procedure require the absence of any definition for abducibles . However, when representing incomplete information, it is often the case that for some predicate a partial definition is available, expressing known information about that predicate. In this case, we can apply a transformation to T so that the resulting program T' has no definition for abducible predicates. This is done by introducing an auxiliary predicate δ_a/n for each abducible predicate a/n with a partial definition and by adding the clause

$$a(\vec{X}) \leftarrow \delta_a(\vec{X})$$

Predicate a/n is no longer abducible , whereas δ_a/n is now abducible. If $a(\vec{t})$ cannot be derived using the partial definition for a/n, it can be derived by abducing $\delta_a(\vec{t})$, provided that it is consistent with integrity constraints.

Usually, observations to be explained are positive. However, by representing negation by default through abduction, we are able to explain also negative observations. A negative observation is represented by a literal *not_l* and an explanation for it can be generated by the abductive proof procedure applied to the goal $\leftarrow$ *not_l*. The explanation of a negative observation has the following meaning: if all the atoms in the explanation for $\leftarrow$ *not_l* are added to the theory, then $\leftarrow l$ will not be derivable. This differs from abductive frameworks proposed by Abe in this volume for whom the explanation of negative observations is uncommon, and by Sakama (this volume, too) for whom the explanation of negative observations is not allowed.

1.2.2 Inductive Logic Programming

The ILP problem can be defined as (Bergadano and Gunetti, 1996) :
Given:

- a set E^+ of positive examples (atoms)
- a set E^- of negative examples (atoms)
- a logic program B (*background knowledge*)
- a set $\mathcal{P}$ of possible programs

Find:

- a logic program $P \in \mathcal{P}$ such that
  - $\forall e^+ \in E^+$, $B \cup P \vdash e^+$ (*completeness*)
  - $\forall e^- \in E^-$, $B \cup P \nvdash e^-$ (*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 learned program P *covers* an example e if $P \cup B \vdash e$. A theory that covers all positive examples is said to be *complete* while a theory that does not cover any negative example is said to be *consistent*. The set $\mathcal{P}$ is called the *hypothesis space*.

The *language bias* (or simply *bias* in this paper) is a description of the hypothesis space. Some systems require an explicit definition of this space and many formalisms have been introduced in order to describe it (Bergadano and Gunetti, 1996). In order to ease the implementation of the algorithm, we have considered only a very simple bias in the form of a set of literals which are allowed in the body of clauses for target predicates.

1.2.3 The New Learning Framework

We consider a new learning problem where both background and target theory are abductive theories and the notion of deductive coverage is replaced by abductive coverage.

Let us first define the *correctness* of an abductive logic program T with respect to the training set E^+, E^-. This notion replaces those of completeness and consistency for logic programs.

Definition 1 (Correctness) *An abductive logic program T is* correct*, with respect to E^+ and E^-, iff there exists Δ such that*

$$T \vdash_{\emptyset}^{\Delta} E^+, not\_E^-$$

where $not\_E^- = \{not\_e^- | e^- \in E^-\}$ and E^+, not_E^- stands for the conjunction of each atom in E^+ and not_E^-

Definition 2 (Abductive Learning Problem)
Given:

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

Find:

A new abductive theory $T' = \langle P \cup P', A, IC \rangle$ such that $P' \in \mathcal{P}$ and T' is correct wrt. E^+ and E^-.

We say that a positive example e^+ is *covered* if $T \vdash_{\emptyset}^{\Delta} e^+$. We say that a negative example e^- is *not covered* (or *ruled out*) if $T \vdash_{\emptyset}^{\Delta} not\_e^-$

The abductive program that is learned can contain new rules (possibly containing abducibles in the body), but not new abducible predicates[2] and new integrity constraints.

We now give an example of an Abductive Learning Problem.

Example 1 *We want to learn a definition for the concept* $father$ *from a background knowledge containing facts about the concepts* $parent$, $male$ *and* $female$. *Knowledge about* $male$ *and* $female$ *is incomplete and we can make assumptions about them by considering them as an abducible.*

Consider the following training sets and background knowledge:

$$
\begin{array}{ll}
E^+ = & \{father(john, mary), father(david, steve)\} \\
E^- = & \{father(john, steve), father(kathy, ellen)\} \\
P = & \{parent(john, mary), male(john), \\
 & parent(david, steve), \\
 & parent(kathy, ellen), female(kathy)\} \\
A = & \{male/1, female/1\} \\
IC = & \{\leftarrow male(X), female(X)\}
\end{array}
$$

Moreover, let the bias be

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

A solution to this Abductive Learning Problem is the theory $T' = \langle P \cup P', A, IC \rangle$ *where*

$$P' = \{father(X, Y) \leftarrow parent(X, Y), male(X)\}$$

In fact, the condition on the solution

$$T \vdash_{\emptyset}^{\Delta} E^+, not\_E^-$$

is verified with

$$\Delta = \{male(david), not\_female(david), not\_male(kathy)\}.$$

Note that, for the example $father(david, steve)$, the abductive proof procedure returns the explanation $\{male(david), not\_female(david)\}$ containing also the literal $not\_female(david)$ which is implied by the constraints and $male(david)$. However, in this way the explanation is such that it can not be consistently extended without violating the constraints.

Differently from the ILP problem, we require the conjunction of examples to be derivable, instead of each example singularly. This is done in order to avoid that abductive explanations for different examples are inconsistent with each other, as it is shown in the next example.

[2] If we exclude the abducible predicates added in order to deal with exceptions, as explained in section 1.3.

procedure LearnAbdLP(
  inputs : E^+, E^- : training sets,
    $T = \langle P, A, IC \rangle$: background abductive theory,
  outputs : P' : learned theory, Δ : abduced literals)

$P' := \emptyset$
$\Delta := \emptyset$
repeat (covering loop)
  GenerateRule(in: T, E^+, E^-, P', Δ; out: $Rule, E^+_{Rule}, \Delta_{Rule}$)
  Add to E^+ all the positive literals of target predicates in Δ_{Rule}
  Add to E^- all the atoms corresponding to
    negative literals of target predicates in Δ_{Rule}
  $E^+ := E^+ - E^+_{Rule}$
  $P' := P' \cup \{Rule\}$
  $\Delta := \Delta \cup \Delta_{Rule}$
until $E^+ = \emptyset$ (Completeness stopping criterion)
output P'

Figure 1.1 The covering loop

Example 2 *Consider the following abductive theory:*

$P = \{p \leftarrow a.$
$\quad q \leftarrow b.\}$
$A = \{a/0, b/0\}$
$IC = \{\leftarrow a, b.\}$

and consider two positive examples p and q. If taken singularly, they are both abductively derivable from the theory with, respectively, the explanations $\{a\}$ and $\{b\}$. However, these explanations are inconsistent with each other because of the integrity constraint $\leftarrow a, b$, therefore the conjunction p, q will not be abductively derivable in the theory.

1.3 AN ALGORITHM FOR LEARNING ABDUCTIVE LOGIC PROGRAMS

In this section, we present an algorithm that is able to learn abductive logic programs according to definition 2. It evolved from the one we proposed in (Esposito et al., 1996).

The algorithm is obtained from the basic top-down ILP algorithm (Bergadano and Gunetti, 1996), by replacing, for the coverage test of examples, the Prolog proof procedure with the abductive proof procedure.

As the basic one, our algorithm is constituted by two nested loops: the covering loop (Figure 1.1) and the specialization loop (Figure 1.2). 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. Positive examples covered by the rule are removed from the training set and a new iteration of the covering

procedure GenerateRule(
  inputs : T, E^+, E^-, P', Δ
  outputs : $Rule$: rule,
    E^+_{Rule} : positive examples covered by $Rule$,
    Δ_{Rule} : abduced literals

Select a target predicate p
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(in: $Rule, T, P', E^+, E^-, \Delta$,
    out: $E^+_{Rule}, E^-_{Rule}, \Delta_{Rule}$)
  if $E^+_{Rule} = \emptyset$
    backtrack to a different choice for L
until $E^-_{Rule} = \emptyset$ (Consistency stopping criterion)
output $Rule, E^+_{Rule}, \Delta_{Rule}$

Figure 1.2 The specialization loop

procedure TestCoverage(
  inputs : $Rule, T, P', E^+, E^-, \Delta$
  outputs: E^+_{Rule}, E^-_{Rule}: examples covered by $Rule$
    Δ_{Rule} : new set of abduced literals

$E^+_{Rule} := E^-_{Rule} := \emptyset$
$\Delta_{in} := \Delta$
for each $e^+ \in E^+$ **do**
  if AbdDer($\leftarrow e^+, \langle P \cup P' \cup \{Rule\}, A, IC\rangle, \Delta_{in}, \Delta_{out}$)
    succeeds **then** Add e^+ to E^+_{Rule}; $\Delta_{in} := \Delta_{out}$
endfor
for each $e^- \in E^-$ **do**
  if AbdDer($\leftarrow not\_e^-, \langle P \cup P' \cup \{Rule\}, A, IC\rangle, \Delta_{in}, \Delta_{out}$)
    succeeds **then** $\Delta_{in} := \Delta_{out}$
  else Add e^- to E^-_{Rule}
endfor
$\Delta_{Rule} := \Delta_{out} - \Delta$
output $E^+_{Rule}, E^-_{Rule}, \Delta_{Rule}$

Figure 1.3 Coverage testing

loop is started. The algorithm ends when the set of positive examples becomes empty. The new clause is generated in the specialization loop: the clause is initially assigned an empty body, and literals are added to it 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 1.3), an abductive derivation is started for each positive example. This derivation results in a (possibly empty) set of abduced literals. We give as input to the abductive procedure also the set of literals abduced in the derivations of previous examples. In this way, we ensure that assumptions made during the derivation of the current example are consistent with assumptions for other examples.

Second, in order to check that no negative example is covered ($E^-_{Rule} = \emptyset$ in Figure 1.2) by the generated rule $Rule$, an abductive derivation is started for the default negation of each negative example ($\leftarrow not\_e^-$). Also in this case, each derivation starts from the set of abducibles previously assumed. The set of assumptions is initialized to the empty set at the beginning of the computation, and is gradually extended as it is passed on from derivation to derivation. This is done as well across different clauses.

Third, some abducible predicates may be also target predicates, i.e., predicates for which we want to learn a definition. To this purpose, after the generation of each clause, abduced atoms of target predicates are added to the training set, so that they become new training examples. For each positive abduced literal l, if it is positive, l is added to E^+, if it is negative, $\overline{l}$ is added to E^-.

In order to achieve consistency, in rule specialization (Figure 1.2), a rule can be specialized by adding either a non-abducible literal or an abducible one. However, the system does not need to be aware of what kind of literal it is adding to the rule since the abductive proof procedure takes care of both cases. When adding an abducible atom $\delta(\vec{X})$ that has no definition in the background, the rule becomes consistent because each negative example $p(\vec{t^-})$ is uncovered by assuming $not\_\delta(\vec{t^-})$ and each previously covered positive example $p(\vec{t^+})$ is still covered by assuming $\delta(\vec{t^+})$. If the abducible has a partial definition, some positive examples will be covered without abduction and others with abduction, while some negative examples will be uncovered with abduction and others will be covered.

We prefer to first try adding non-abducible literals to the rule, since complete information is available about them and therefore the coverage of examples is more certain.

The algorithm also performs the task of learning exceptions to rules. The task of learning exceptions to rules is a difficult one because exceptions limit the generality of the rules since they represent specific cases. In order to deal with exceptions, a number of (new) auxiliary abducible predicates is provided, so that the system can use them, for rule specialization, when no standard

literal or abducible literal with a partial definition is available from the bias such that a rule for a target predicate becomes consistent.

The algorithm can be extended in order to learn not only from examples but also from integrity constraints on target predicates. The details of this extension together with an example application will be described in section 1.4.4.

Note that the system is not able to learn full abductive theories, including new integrity constraints as well. In order to do this, in (Kakas and Riguzzi, 1997) the authors proposed the use of systems that learn from interpretations, such as Claudien (De Raedt and Bruynooghe, 1993) and ICL (De Raedt and Van Laer, 1995).

1.4 EXAMPLES

Two interesting applications of the integration are learning from incomplete knowledge and learning exceptions. When learning from incomplete knowledge, abduction completes the information available in the background knowledge. When learning exceptions, instead, assumptions are used as new training examples in order to generate a definition for the class of exceptions (section 1.4.3).

When learning from incomplete data, what to do with the assumptions depends on the type of theory we are learning. When learning a non-abductive theory, abduction completes the information available in the background knowledge and it is therefore natural to add the assumptions to the theory at the end of the learning process, thus doing a form of theory revision (section 1.4.1). When learning an abductive theory, the assumptions made do not have to be added to the theory, since they can be guessed by abduction in the final theory (section 1.4.2).

1.4.1 Learning from incomplete knowledge

Abduction is particularly suitable for modelling domains in which there is incomplete knowledge. In this section, we show how the algorithm is able to find the solution of the learning problem presented in example 1.

For the sake of clarity, in the following we repeat the problem statement. We want to learn a definition for the concept $father$ from a background knowledge containing facts about the concepts *parent*, *male* and $female$, with *male* and $female$ being incompletely defined.

Consider the abductive background theory $B = \langle P, A, IC \rangle$ and training set:

$$
\begin{aligned}
P = \; & \{parent(john, mary), male(john), \\
& parent(david, steve), \\
& parent(kathy, ellen), female(kathy)\} \\
A = \; & \{male/1, female/1\} \\
IC = \; & \{\leftarrow male(X), female(X)\} \\
E^+ = \; & \{father(john, mary), father(david, steve)\} \\
E^- = \; & \{father(john, steve), father(kathy, ellen)\}
\end{aligned}
$$

Moreover, let the bias be

$father(X,Y) \leftarrow \alpha$ where $\alpha \subset \{parent(X,Y), parent(Y,X),$
$male(X), male(Y), female(X), female(Y)\}$

The program must first be transformed into its positive version and then into a program where abducibles have no definition, as shown in section 1.2.1. For simplicity, we omit the two transformations, and we suppose to apply the inverse transformations to the learned program.

At the first iteration of the specialization loop, the algorithm generates the rule

$father(X,Y) \leftarrow$.

which covers all positive examples but also all 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 positive examples but also the negative example $father(kathy, ellen)$. 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 abduction, while $father(david, steve)$ is covered with the abduction of $\{male(david), not\_female(david)\}$.

Then the coverage of negative examples is tested by starting the abductive derivations

$\leftarrow not\_father(john, steve)$.

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

The first derivation succeeds with an empty explanation while the second succeeds abducing $not\_male(kathy)$ which is consistent with the fact $female(kathy)$ and the constraint $\leftarrow male(X), female(X)$. Now, no negative example is covered, therefore the specialization loop ends. No target atom is in Δ, therefore no example is added to the training set. 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, returning the rule

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

and the assumptions

$\Delta = \{male(david), not\_female(david), not\_male(kathy)\}$.

At this point, assumptions made are added to the background knowledge in order to complete the theory, thus performing a kind of theory revision. Only positive assumptions are added to the resulting theory, since negative assumptions can be derived by Negation As Failure[3]. In this case, only $male(david) \leftarrow$. is added to the theory, while *not* $female(david)$ and *not* $male(kathy)$ can be derived by using Negation As Failure.

[3]Since Prolog proof procedure will be used with the final theory and therefore Negation As Failure will replace Negation by Default.

1.4.2 Learning Abductive Theories

In this section, we apply the algorithm to the problem of learning an abductive theory for diagnosis from an incomplete background knowledge. When learning an abductive theory, the observations to be explained are represented by facts in the training set and the corresponding known explanations by facts in the background knowledge.

It must be observed that, when the available information on explanations is complete, it is not necessary to use our algorithm for learning the rule part of the theory, but any standard ILP system can be used. In fact, if all the explanations are known, then every positive example can be entailed by the resulting theory without the need of abduction. Therefore, we argue that the main use of abduction in learning is for completing incomplete knowledge.

In presence of incomplete information on the explanations, abduction is necessary to generate the missing (part of) explanations, as in the general case of learning from incomplete knowledge (see section 1.4.1). However, differently from that case, explanations should not be added to the resulting theory. In fact, explanations can be newly obtained from the target theory by means of abductive reasoning.

Let us consider the case of an abductive background theory containing the following clauses, abducibles and constraints:

$$\begin{aligned} P = \quad & \{flat\_tyre(bike_1). \\ & circular(bike_1). \\ & tyre\_holds\_air(bike_3). \\ & circular(bike_4). \\ & tyre\_holds\_air(bike_4).\} \end{aligned}$$

$$A = \quad \{flat\_tyre/1, broken\_spokes/1\}$$

$$\begin{aligned} IC = \quad & \{\leftarrow flat\_tyre(X), tyre\_holds\_air(X). \\ & \leftarrow circular(X), broken\_spokes(X).\} \end{aligned}$$

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

The algorithm generates the following clause in the specializing loop:

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

Then the clause is tested. This clause covers $wobbly\_wheel(bike_1)$ because $flat\_tyre(bike_1)$ is specified in the background knowledge and it covers $wobbly\_wheel(bike_2)$ by assuming

$$\{flat\_tyre(bike_2), not\_tyre\_holds\_air(bike_2)\}.$$

The example $wobbly\_wheel(bike_3)$, however, cannot be covered: in fact, we cannot assume $flat\_tyre(bike_3)$ since it is inconsistent with the integrity constraint

$\leftarrow flat\_tyre(X), tyre\_holds\_air(X)$. and the fact $tyre\_holds\_air(bike_3)$. Then, we check that $not\_wobbly\_wheel(bike_4)$ is derivable in the hypotheses set. This derivation succeeds by abducing $not\_flat\_tyre(bike_4)$.

The algorithm adds the clause to the current theory and removes covered examples from E^+. A new iteration of the covering loop is then started with:

$$E^+ = \{wobbly\_wheel(bike_3)\},$$
$$E^- = \{wobbly\_wheel(bike_4)\}$$
$$\Delta = \{flat\_tyre(bike_2), not\_tyre\_holds\_air(bike_2), not\_flat\_tyre(bike_4)\}.$$

In order to cover the remaining positive example $wobbly\_wheel(bike_3)$, the system generates the clause:

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

which covers the example by abducing

$$\{broken\_spokes(bike_3), not\_circular(bike_3)\}$$

In fact, these assumptions are consistent with the integrity constraint:

$$\leftarrow circular(X), broken\_spokes(X).$$

As for the previous case, the negative example is ruled out by assuming $not\_broken\_spokes(bike_4)$. At this point the algorithm terminates because E^+ becomes empty.

The resulting set of assumptions constitute a set of diagnosis for the devices considered in the training set. Assumptions are not added to the resulting theory since they can be generated by abductive reasoning at any time.

1.4.3 Learning Rules with Exceptions

The task of learning exceptions to rules is a difficult one because exceptions limit the generality of the rules since they represent specific cases. In the following, we discuss how our algorithm performs the task of learning exceptions to rules by using a number of auxiliary abducible predicates, and show an example of its behaviour.

In order to learn exceptions to rules, when no standard literal or abducible literal with a partial definition is available from the bias such that a rule for a target predicate p/n becomes consistent, then the algorithm specializes the rule by adding a new abducible literal $not\_abnorm_i(\vec{X})$. This addition transforms the rule into a default rule that can be applied in all "normal" (or non-abnormal) cases. The refined rule becomes consistent by abducing $abnorm_i(\vec{t^-})$ for every negative example $p(\vec{t^-})$. Positive examples, instead, will be covered by abducing $not\_abnorm_i(\vec{t^+})$ for every positive example $p(\vec{t^+})$. These assumptions are then added to the training set, and are used to learn a definition for $abnorm_i/n$ that describes the class of exceptions. If there are exceptions to

exceptions, the system adds a new literal not_abnorm_j/n to the body of the rule for $abnorm_i/n$ and the process is iterated. Therefore, we are able to learn hierarchies of exceptions.

The above technique is implemented by including a number of predicates of the form not_abnorm_i/n in the bias of each target predicate of p/n that may have exceptions. Moreover, $abnorm_i/n$ and not_abnorm_i/n are added to the set of abducible predicates and the constraint

$$\leftarrow abnorm_i(\vec{X}), not\_abnorm_i(\vec{X}).$$

is added to the (positive version of the) background knowledge. Predicates $abnorm_i/n$ are considered as target predicates, and a bias must be defined for them. Since we may have exceptions to exceptions, we may also include a number of literals of the form $not\_abnorm_j(\vec{X})$ in the bias for $abnorm_i/n$.

The example which follows is inspired by (Dimopoulos and Kakas, 1995), and shows how exceptions are dealt with. Let us consider the following background abductive theory $T = \langle P, A, IC \rangle$ and training sets:

$P = \{bird(X) \leftarrow penguin(X).$
$\quad penguin(X) \leftarrow superpenguin(X).$
$\quad bird(a).$
$\quad bird(b).$
$\quad penguin(c).$
$\quad penguin(d).$
$\quad superpenguin(e).$
$\quad superpenguin(f).\}$
$A = \{abnorm_1/1, abnorm_2/1\}$
$IC = \{\}$

$E^+ = \{flies(a), flies(b), flies(e), flies(f)\}$
$E^- = \{flies(c), flies(d)\}$

The positive version of the theory will contain also the constraints:

$$\leftarrow abnorm_1(X), not\_abnorm_1(X).$$
$$\leftarrow abnorm_2(X), not\_abnorm_2(X).$$

Moreover, let the bias be:

$flies(X) \leftarrow \alpha$ where
$\alpha \subseteq \{bird(X), penguin(X), superpenguin(X),$
$\quad not\_abnorm_1(X)\}$
$abnorm_1(X) \leftarrow \beta$ where
$\beta \subseteq \{bird(X), penguin(X), superpenguin(X),$
$\quad not\_abnorm_2(X)\}$
$abnorm_2(X) \leftarrow \gamma$ where
$\gamma \subseteq \{bird(X), penguin(X), superpenguin(X)\}$

The algorithm starts by generating the following rule in the specialization loop (R_1):

$flies(X) \leftarrow bird(X).$

which covers all positive examples, but also all negative ones. In order to rule out negative examples, the abducible literal not_abnorm_1 is added to the body of R_1 obtaining R_2:

$flies(X) \leftarrow bird(X), not\_abnorm_1(X).$

Now, the theory is correct and the set of assumptions resulting from the derivations of positive and (negated) negative examples is

$\{not\_abnorm_1(a), not\_abnorm_1(b),$
$not\_abnorm_1(e), not\_abnorm_1(f),$
$abnorm_1(c), abnorm_1(d)\}.$

Since $abnorm_1/1$ is a target predicate, these assumptions become new training examples yielding:

$E^+ = \{abnorm_1(c), abnorm_1(d)\}$
$E^- = \{abnorm_1(a), abnorm_1(b), abnorm_1(e), abnorm_1(f)\}$

Therefore, a new iteration of the covering loop is started in which the following clause is generated (R_3):

$abnorm_1(X) \leftarrow penguin(X), not\_abnorm_2(X).$

The rule is correct and the set of assumptions resulting from the derivations of positive and (negated) negative examples is

$\{not\_abnorm_2(c), not\_abnorm_2(d),$
$abnorm_2(e), abnorm_2(f)\}.$

Note that no assumptions are generated for the derivations

$\leftarrow not\_abnorm_1(a)$
$\leftarrow not\_abnorm_1(b)$

since $penguin(a)$ and $penguin(b)$ are false.

After the addition of the new assumptions, the training sets become

$E^+ = \{abnorm_2(e), abnorm_2(f)\}$
$E^- = \{abnorm_2(c), abnorm_2(d)\}$

For this training set, the algorithm produces the rule

$abnorm_2(X) \leftarrow superpenguin(X).$

that is correct and no assumption is generated for covering examples. The algorithm now ends by producing the following program:

$flies(X) \leftarrow bird(X), not\_abnorm_1(X).$
$abnorm_1(X) \leftarrow penguin(X), not\_abnorm_2(X).$
$abnorm_2(X) \leftarrow superpenguin(X).$

We try to generalize exceptions in order to treat them as a whole, possibly leading to the discovery of exceptions to exceptions. In this way, we can learn hierarchies of exceptions.

1.4.4 Learning from Integrity Constraints on Target Predicates

We now present a way in which training examples can be extracted from integrity constraints on target predicates. In this way, the algorithm is able to learn not only from examples but also from integrity constraints, as it is done in (De Raedt and Bruynooghe, 1992; Adé et al., 1994; Muggleton, 1995).

Let us consider the abductive program T' generated in the previous section, and add to it the following user-defined constraint, I, on target predicates:

$$\leftarrow rests(X), plays(X).$$

Consider now the new training sets:

$$E^+ = \{plays(a), plays(b), rests(e), rests(f)\}$$
$$E^- = \{\}$$

In this case, the information about the target predicates comes not only from the training set, but also from integrity constraints. These constraints contain target predicates and therefore they differ from those usually given in the background knowledge that contain only non-target predicates, either abducible or non-abducible. The generalization process is not limited by negative examples but by integrity constraints. Suppose that we generalize the two positive examples for $plays/1$ in $plays(X)$. This means that for all X, $plays(X)$ is true. However, this is inconsistent with the integrity constraint I because $plays(X)$ cannot be true for e and f.

The information contained in these type of integrity constraints must be made available in a form that is exploitable by our learning algorithm, i.e., it must be transformed into new training examples, as it is done in theory revision systems (De Raedt and Bruynooghe, 1992; Adé et al., 1994). When the knowledge base violates a newly supplied integrity constraint, these systems extract one example from the constraint and revise the theory on the basis of it: in (De Raedt and Bruynooghe, 1992) the example is extracted by querying the user on the truth value of the literals in the constraint, while in (Adé et al., 1994) the example is automatically selected by the system.

In our approach, one or more examples are generated from constraints on target predicates using the abductive proof procedure. The consistency of each available example is checked with the constraints, and assumptions are possibly made to ensure consistency. Assumptions about target predicates are considered as new negative or positive examples.

In the previous case, we start an abductive derivation for

$$\leftarrow plays(a), plays(b), rests(e), rests(f)$$

Since $plays/1$ and $rests/1$ are abducibles, a consistency derivation is started for each atom. Consider $plays(a)$, in order to have the consistency with the constraint $\leftarrow plays(X), rests(X).$, the literal $not\_rests(a)$ is abduced. The same is done for the other literals in the goal obtaining the set of assumptions

$$\{not\_rests(a), not\_rests(b), not\_plays(e), not\_plays(f)\}$$

that is then transformed in the set of negative examples

$$E^- = \{rests(a), rests(b), plays(e), plays(f)\}$$

Now the learning process applied to the new training set generates the following correct rules:

$$plays(X) \leftarrow bird(X), not\_abnorm_1(X).$$
$$rests(X) \leftarrow superpenguin(X).$$

In this way, we can learn not only from (positive and negative) examples but also from integrity constraints.

1.5 INTEGRATION OF ABDUCTION AND INDUCTION

In this section, we describe our approach for the integration of abduction and induction, and we relate it to other works.

In our approach, abduction is used in induction for making assumptions about unknown facts in order to cover examples, as proposed by Abe, Sakama, Mooney (this volume) and (Dimopoulos and Kakas, 1996; De Raedt and Bruynooghe, 1992; Adé et al., 1994; Adé and Denecker, 1995; Kanai and Kunifuji, 1997).

Abducibles can be present in the body of background rules or can be added to the body of a target rule for specialization. These assumptions can be relative to background abducible predicates, with an empty or partial definition, or to target predicates, for which a definition must be learned, as in work by Mooney (this volume) and in (De Raedt and Bruynooghe, 1992; Adé et al., 1994; Adé and Denecker, 1995; Kanai and Kunifuji, 1997). In this second case, assumptions are also added to the training set so that they become new training examples. Induction is then used in order to generalize assumptions. In both cases, the set of assumptions is stored and gradually extended in order to ensure consistency among examples. At the end of the computation, the resulting set of assumptions can be discarded, if we are learning an abductive theory, or added to the theory, if we want to complete an incomplete theory.

In this way, we obtain a particular instantiation of the cycle of abductive and inductive knowledge development described by Flach and Kakas in this volume. In our approach, abduction helps induction by generating suitable background knowledge or training examples, while induction helps abduction by generalizing the assumptions made.

Abe, Sakama, Mooney (this volume) and (Dimopoulos and Kakas, 1996; De Raedt and Bruynooghe, 1992; Adé et al., 1994; Adé and Denecker, 1995; Kanai and Kunifuji, 1997) agree on using abduction for covering examples. What to do with the assumptions generated depends then on the task you are performing. If you are performing theory revision, you can revise overspecific rules by dropping the abducible literals from the body of rules (Mooney, Sakama, this volume) or by adding the explanations to the theory (De Raedt and Bruynooghe, 1992; Adé et al., 1994). In both theory revision and batch learning, you can learn a definition for the abducible predicates by considering the

assumptions as examples for the abducible predicates (Mooney, this volume) and (Kanai and Kunifuji, 1997; De Raedt and Bruynooghe, 1992; Adé et al., 1994; Adé and Denecker, 1995).

Another approach for the use of abduction in learning is described in (Dimopoulos and Kakas, 1996) where each example is given together with a set of observations that are related to it. Abduction is then used to explain the observations in order to generate relevant background data for the inductive generalization.

Different positions exist on the treatment of negative observations. According to Abe (this volume), "it is very rare for hypotheses to be generated when observations are negative", therefore he does not consider this possibility. To avoid the difficulties of learning from positive examples only, he adopts Abductive Analogical Reasoning to generate abductive hypotheses under similar observations: in this case, generated hypotheses satisfy the Subset Principle and it is possible to learn from positive examples only.

Sakama (this volume), instead, revises a database that covers negative observations by revising only the extensional part of the database: by abduction, he finds the facts that are responsible for the coverage of the negative example/observation and he replaces each such fact $\Sigma \leftarrow$ with the clause $\Sigma \leftarrow \Sigma$. Semantically, $\Sigma \leftarrow \Sigma$ (that is equivalent to $\Sigma \vee \neg\Sigma$, represents two possible worlds, one in which Σ is true and the other in which Σ is false. In this way the inconsistency is removed but, differently from systems where the theory is revised by removing Σ, information about the previous state is kept, thus allowing to restore the database in its original state.

Our approach for the treatment of negative examples is similar to work by Mooney (this volume) and (Kanai and Kunifuji, 1997). It differs from Abe's one (this volume), since we do generate explanations for negative examples, and is a generalization of Sakama's one since we are able to revise not only facts but also rules. The kind of revision we perform is very similar: instead of adding an abnormality literal in the head of the fact, we add a non-abnormality literal to the body. Moreover, Sakama's procedure is effective for dealing with single exceptions rather then classes of exceptions, because it treats each exception singularly by adding a new abducible literal to a fact. Instead, we try to generalize exceptions in order to treat them as a whole, possibly leading to the discovery of a hierarchy of exceptions.

Christiansen (this volume) proposes a reversible *demo* predicate that is able to generate the (parts of the) program that are necessary for deriving the goal. Constraint Logic Programming techniques are used for specifying conditions on the missing program parts in a declarative way. The approach shows a high generality, being able to perform either induction or abduction depending on which program part is missing, general rules or specific facts. The author also shows that the system is able to learn exceptions to rules, though not hierarchies of exceptions.

1.6 CONCLUSIONS AND FUTURE WORK

We have proposed an algorithm where abduction and induction cooperate in order to improve their power. Abduction helps induction by generating suitable background knowledge or training examples, while induction helps abduction by generalizing the assumptions made.

The algorithm solves an extended ILP problem in which both the background and target theories are abductive theories, and coverage by deduction is replaced with coverage by abduction. The algorithm is obtained from the basic top-down ILP algorithm by substituting, for the coverage testing, Prolog proof procedure with an abductive proof procedure. It can be applied to the problems of learning from incomplete knowledge, learning abductive theories and learning rules with exceptions.

Future work will be devoted to evaluate the applicability of these ideas on real world problems and to extend the system for learning full abductive theories, including as well integrity constraints. The integration of the algorithm with other systems for learning constraints, such as Claudien (De Raedt and Bruynooghe, 1993) and ICL (De Raedt and Van Laer, 1995), as proposed in (Kakas and Riguzzi, 1997), seems very promising in this respect.

Another interesting future line of research consists in investigating the idea, proposed by Inoue and Haneda (this volume) of learning logic programs with abduction and two kinds of negation (e.g., default negation and explicit negation).

Appendix

In the following we recall the abductive proof procedure used by our algorithm. The procedure is taken from (Kakas and Mancarella, 1990). It is composed by two phases: abductive derivation and consistency derivation.

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

(A1) If L_j is not abducible or default, then $G_{i+1} = C$ and $\Delta_{i+1} = \Delta_i$ where C is the resolvent of some clause in P with G_i on the selected literal L_j;

(A2) If L_j is abducible or default and $L_j \in \Delta_i$ then
$G_{i+1} = \leftarrow L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_k$ and $\Delta_{i+1} = \Delta_i$;

(A3) 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 $G_{i+1} = \leftarrow L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_k$ and $\Delta_{i+1} = \Delta'$.

Steps $(A1)$ and $(A2)$ are SLD-resolution steps with the rules of P and abductive or default hypotheses, respectively. In step $(A3)$ 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

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

where :

(Ci) F_1 is the union of all goals of the form $\leftarrow L_1, \ldots, L_n$ obtained by resolving the abducible or default α with the denials in IC with no such goal been empty, $\leftarrow$;

(Cii) for each $i > 1$, F_i has the form $\{\leftarrow L_1, \ldots, L_k\} \cup F_i'$ and for some $j = 1, \ldots, k$ $(F_{i+1}\ \Delta_{i+1})$ is obtained according to one of the following rules:

- (C1) If L_j is not abducible or default, then $F_{i+1} = C' \cup F_i'$ where C' is the set of all resolvents of clauses in P with $\leftarrow L_1, \ldots, L_k$ on the literal L_j and $\leftarrow \notin C'$, and $\Delta_{i+1} = \Delta_i$;
- (C2) If L_j is abducible or default, $L_j \in \Delta_i$ and $k > 1$, then $F_{i+1} = \{\leftarrow L_1, \ldots, L_{j-1}, L_{j+1}, \ldots, L_k\} \cup F_i'$ 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 $\leftarrow L_1, \ldots, L_k$ 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. 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, 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.

References

Adé, H. and Denecker, M. (1995). AILP: abductive inductive logic programming. In *Proc. 14th Int'l Joint Conf. on Artificial Intelligence*, pages 1201–1207, California. Morgan Kaufmann.

Adé, H., Malfait, B., and Denecker, M. (1994). RUTH: an ILP theory revision system. In *Proc. 8th Int'l Symp. on Methodologies for Intelligent Systems, Lecture Notes in Artificial Intelligence 869*, pages 336–345. Springer-Verlag, Berlin.

Bergadano, F. and Gunetti, D. (1996). *Inductive Logic Programming*. MIT Press.

Brogi, A., Lamma, E., Mancarella, P., and Mello, P. (1997). A unifying view for logic programming with non-monotonic reasoning. *Theoretical Computer Science*, 184:1–59.

Clark, K. L. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Proceedings of the Symposium on Logic and Databases*, pages 293–322. Plenum Press, New York.

De Raedt, L., editor (1996). *Advances in Inductive Logic Programming*. IOS Press, Amsterdam.

De Raedt, L. and Bruynooghe, M. (1992). Belief updating from integrity constraints and queries. *Artificial Intelligence*, 53:291–307.

De Raedt, L. and Bruynooghe, M. (1993). A theory of clausal discovery. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 1058–1063, Chambéry, France.

De Raedt, L. and Van Laer, W. (1995). Inductive constraint logic. In *Proceedings of the 5th International Workshop on Algorithmic Learning Theory*.

Dimopoulos, Y. and Kakas, A. C. (1995). Learning non-monotonic logic programs: learning exceptions. In Lavrač, N. and Wrobel, S., editors, *Proceedings of the 8th European Conference on Machine Learning, Lecture Notes in Artificial Intelligence, Vol. 912*, pages 122–137. Springer.

Dimopoulos, Y. and Kakas, A. C. (1996). Abduction and inductive learning. In (De Raedt, 1996), pages 144–171.

Eshghi, K. and Kowalski, R. A. (1989). Abduction compared with negation by failure. In Levi, G. and Martelli, M., editors, *Proceedings of the 6th International Conference on Logic Programming*, pages 234–254. MIT Press.

Esposito, F., Lamma, E., Malerba, D., Mello, P., Milano, M., Riguzzi, F., and Semeraro, G. (1996). Learning abductive logic programs. In Flach, P. A. and Kakas, A. C., editors, *Proceedings of the ECAI'96 Workshop on Abductive and Inductive Reasoning*, pages 23–30. Available on-line at `http://www.cs.bris.ac.uk/~flach/ECAI96/`.

Kakas, A. C., Kowalski, R. A., and Toni, F. (1997). The role of abduction in logic programming. In Gabbay, D., Hogger, C., and Robinson, J., editors, *Handbook of Logic in AI and Logic Programming*, volume 5, pages 233–306. Oxford University Press.

Kakas, A. C. and Mancarella, P. (1990). On the relation between truth maintenance and abduction. In *Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence*.

Kakas, A. C. and Riguzzi, F. (1997). Learning with abduction. In Lavrač, N. and Džeroski, S., editors, *Proceedings of the 7th International Workshop*

on Inductive Logic Programming, Lecture Notes in Artificial Intelligence, Vol. 1297, pages 181–188. Springer.

Kanai, T. and Kunifuji, S. (1997). Extending inductive generalisation with abduction. In Flach, P. A. and Kakas, A. C., editors, *Proceedings of the IJCAI'97 Workshop on Abduction and Induction in Artificial Intelligence*, pages 25–30. Available on-line at `http://www.cs.bris.ac.uk/~flach/IJCAI97/`.

Lamma, E., Mello, P., Milano, M., and Riguzzi, F. (1997). A system for learning abductive logic programs. In J. Dix, L. M. Pereira, T. P., editor, *Proceedings of the Workshop on Logic Programming and Knowledge Representation LPKR97*. Universität Koblenz-Landau, Technical Report n. 10/97.

Muggleton, S. H., editor (1992). *Inductive Logic Programming*. Academic Press.

Muggleton, S. H. (1995). Inverse entailment and Progol. *New Generation Computing*, 13(3-4):245–286.

Index