# Abductive Concept Learning

A.C. Kakas and F. Riguzzi

January 31, 1997

**Abstract**

We investigate how abduction and induction can be integrated in order to obtain a more powerful learning framework. In particular, we discuss the possible applications of the Abductive Concept Learning framework, an extension of the Inductive Logic Programming learning paradigm to the case in which both the background and the target theory are abductive logic programs. In this framework, we can learn in the presence of incomplete information in the background knowledge and/or in the training set by exploiting the hypothetical reasoning of abduction. We first state the requirements for a system that performs ACL, and then illustrate how such a system could be used to solve some of the problems of extensional and intensional ILP systems. Abductive logic programs are a powerful means of representing concepts: we investigate the different uses of integrity constraints in the target theory. Finally, we present an algorithm for ACL, which performs a depth-first search in the space of clause orderings and a best-first search in the space of clause refinements, together with an appropriate heuristic function.

## 1 Introduction

In this paper we investigate how abduction and induction can be integrated in order to obtain a more powerful learning framework. In particular, we discuss how the Inductive Logic Programming (ILP) learning paradigm [12, 2] can be extended in order to learn abductive logic programs instead of definite or normal logic programs.

The extended inductive problem resulting from this integration was introduced in [6] and was called Abductive Concept Learning.

**Definition 1.1** *Abductive Concept Learning (ACL)*
**Given**

- *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* $AT' = \langle T', A, IC' \rangle$ *such that*

- *for each* $e^+ \in E^+$, $AT' \models_A e^+$,
- *for each* $e^- \in E^-$, $AT' \not\models_A e^-$.

Therefore, ACL differs from ILP because both the background knowledge and the learned program are abductive logic programs. As a consequence, the notion of entailment of ILP must be substituted with the notion of abductive entailment ($\models_A$). ACL is a new learning paradigm that contains ILP as a special case.

In the following, we will consider a modified version of this definition of ACL, in which the last condition is substituted by

- *for each* $e^- \in E^-$, $AT' \models_A$ *not* $e^-$.

We will call the first definition ACL1, while the latter ACL2. When mentioning simply ACL we will be referring to ACL2. In section 2 we will discuss the differences between the two definitions.

ACL can be used when the background theory is available in the form of an abductive logic program. This is usually the case when the background knowledge is incomplete, because we are lacking some information on the domain. In this case, a system for ACL can be used to learn despite the incompleteness. Some of the background predicates will be considered as abducible: these are predicates for which we suspect or we know that they have an incomplete definition. During the learning process, we can make assumptions, by means of abduction, about these predicates, provided that these assumptions are consistent with the available partial definition given in terms of rules and integrity constraints. These assumptions are made in order to provide support for the theory that we are learning.

We will show that ACL is able to deal also with a different type of incompleteness: sparseness of the training set. In this case the target predicates are considered abducible and, during the learning process, a system for ACL tries to complete the training data by using abduction. In this case, the assumptions have to be consistent with the training data and with the partial definition available for the target predicate.

The aim of the learning process is to produce a complete definition of the target predicates despite the incompleteness of the available information. If this is not possible, we could consider them as abducible (or incomplete) in the final theory and try to restrict the assumptions that can be made on them by learning integrity constraints. For abducible predicates in the background knowledge, we would also like to find stricter boundaries by inferring integrity constraints.

Let us illustrate these ideas by means of an example.

**Example 1.2** *Suppose we want to learn the concept* $father$. *Let the background theory be:*

$T = \{parent(john, mary), male(john),$
$parent(david, steve),$
$parent(katy, ellen), female(katy)\}$
$A = \{male, female\}$

*and let the training data be:*

$E^+ = \{father(john, mary), father(david, steve)\}$
$E^- = \{father(katy, ellen)\}$

*In this case, we would like an ACL system to learn the rule*

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

*making the assumptions* $\Delta = \{male(david), not\ male(katy)\}$. *By considering the background knowledge together with these assumptions, we could infer the integrity constraint:*

$\leftarrow male(X), female(X).$

The next example shows that ACL can be particularly useful for performing multiple predicate learning, because assumptions made while learning one concept can be used as training data for learning another concept.

**Example 1.3** *Suppose we want to learn the concepts* $grandfather$ *and* $father$. *Let the background theory be:*

$T = \{parent(john, mary), father(john, mary), male(john),$
$parent(mary, ellen), female(mary),$
$parent(ellen, sue),$
$parent(david, steve),$
$parent(steve, jim)\}$
$A = \{father, male, female\}$

*and let the training data be:*

$E^+ = \{grandfather(john, ellen), grandfather(david, jim)\}$
$E^- = \{grandfather(mary, sue)\}$

*An ACL system should learn the rule:*

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

*abducing* $\Delta_1 = \{father(david, steve), not\ father(mary, ellen)\}$. *Now we learn a definition for father using* $T \cup \Delta_1$*:*

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

*abducing* $\Delta_2 = \{male(david), not\ male(mary)\}$. *Moreover, as in example 1.2, we can infer an integrity constraint from* $T \cup \Delta_1 \cup \Delta_2$*:*

$\leftarrow male(X), female(X).$

In the previous examples, we have seen that integrity constraints are generated for predicates of the background knowledge. However, when the target predicate is abducible as well and we have made assumptions about it in the learning phase, we may want to generate constraints on it, as it is shown in the next example.

**Example 1.4** *Let the background theory be*
$T = \{r(a), q(b,a), q(c,d), s(e)\}$
$A = \{p\}$
*and the training data be*
$E^+ = \{p(a), p(b), p(c)\}$
$E^- = \{p(e)\}$
*In this case, the rules*
$p(X) \leftarrow r(X).$
$p(X) \leftarrow q(X,Y), p(Y).$
*could be learned, abducing* $\Delta = \{p(d)\}$. *At this point we could stop the learning process because we have a complete and consistent program w.r.t. to the training data. But the learned program doe not gives a complete definition of the concept p, because we have made an assumption about it which is not covered by any rule. Therefore p will remain abducible as well in the learned theory, allowing for new assumptions on p. However, we could make the assumption* $p(e)$, *which is false because it is a negative example for p. Therefore we have to generate integrity constraints on p, in order to avoid the assumption of false hypothesis. In this case, the integrity constraint*
$\leftarrow p(X), s(X).$
*could be generated, which prohibits the assumption of* $p(e)$

Let us now summarize the tasks that an algorithm for ACL has to perform, starting from the simpler one and going to the more complex.

(1) Make assumptions about abducible literals from the background knowledge in order to cover positive examples and rule out negative ones.

(2) Make assumptions on target literals again in order to cover positive and rule out negative examples. In this way we enlarge the training set and therefore new learning steps may be required.

(3) Infer integrity constraints on the assumptions made on abducible predicates from the background knowledge, therefore reducing the incompleteness of their definition.

(4) Infer integrity constraints on target predicates. For this case we have to distinguish two different uses of constraints:

   (i) in the case which the rules inferred are correct (they do not derive false facts), integrity constraints can be used to limit the assumptions about the facts that cannot be proved using the rules;

   (ii) in the case in which the rules inferred are not correct, we can use integrity constraints to specialize them in order to avoid the derivation of false facts. In this case, everything that has been derived using the rules must be verified against integrity constraints.

In examples 1.2 and 1.3 the system performs tasks (1) and (3), while in example 1.4 tasks (2) and (4)(i) are performed.

Up to now, two algorithms for ACL have been defined. The one proposed in [8] extends an intensional top-down ILP system [2] with abduction, while the one proposed in [11] extends FOIL [15], an extensional top-down system [2], with abduction. Both of them perform only task (1) above. In this paper we present an algorithm which is able to perform all four tasks.

In section 2 we illustrate the differences between the two definitions of ACL. The problems of extensional systems are discussed in section 3 and it is shown how tasks (1) and (2) can be used to solve some of them. In section 4 we discuss how the same tasks can help intensional systems to overcome some problems when learning multiple predicates and normal logic programs. The relation between rules and integrity constraints in case (4) is discussed in section 5, while in section 6 we concentrate on case (ii). An algorithm that performs all four tasks above is presented in section 7. The heuristics for this algorithm are discussed in 8. In section 9, we show how this algorithm can be extended in order to comply with ACL1. Finally, in section 10, we conclude summarizing the main points analyzed in the paper and discussing future works.

# 2 Differences between the two definitions of ACL

ACL1 requires that the negative examples can not be abductively derived from the learned theory. This means that there must be no way of making assumptions that allow to derive the negative examples. In ACL2 instead, we test each negative example $e^-$ by trying to abductively derive *not* $e^-$. In order to derive *not* $e^-$, we can make assumptions to support the goal. Therefore this condition is weaker than the previous one and it allows us to exploit abduction not only to cover positive examples more easily but also to rule out negative examples. Let us illustrate this difference by means of an example.

**Example 2.1** *Consider example 1.2 and suppose that the integrity constraint*
$\leftarrow male(X), female(X).$
*is in the background knowledge. Suppose also that the fact* $female(katy)$ *is not in the background knowledge. We want to test the negative example* $father(katy, ellen)$ *according to ACL1. The test fails, because it is possible to derive* $father(katy, ellen)$ *by assuming* $male(katy)$*, since the integrity constraint is not violated by this assumption. If we test the example according to ACL2, the test succeeds because we can derive not* $father(katy, ellen)$ *by abducing not* $male(katy)$*.*

*If we consider the same learning problem but with* $female(katy)$ *in the background knowledge, now the derivation of* $father(katy, ellen)$ *fails, and the negative example is not covered also according to ACL1. The derivation of not* $father(katy, ellen)$ *on its turn, succeeds without the abduction of anything.*

If the abductive derivation of a negative example fails, then its negation can be derived without making any assumption. Therefore

$$(AT \not\models_A e^-) \Rightarrow (AT \models_A not\ e^-)$$

The opposite implication is not true, as it has been shown in the example.

# 3 ACL for extensional systems

Extensional coverage [2] has been used instead of intensional coverage in order to partially solve the problems of multiple predicate learning. However, extensionality has introduced other problems, summarized in [16], due to the fact that, during learning, theories are tested in an extensional way, while after learning they are used intensionally, and the behaviour can be different. These problems arise only when we are trying to learn recursive predicates or when we are trying to learn multiple predicates. The possible cases of discrepancies can be summarized as follows:

- Extensional consistency, intensional inconsistency.
- Extensional completeness, intensional incompleteness.
- Intensional completeness, extensional incompleteness.

Let us examine each of them in more detail.

## 3.1 Extensional consistency, intensional inconsistency

This problem can arise in two cases. To illustrate the first one, suppose we are trying to learn a theory for the concepts $p$ and $q$ and we have learned the rule

$$p(X) \leftarrow \ldots q(X)$$

A certain negative example $p(e^-)$ is not covered if $q(e^-)$ is a negative example for $q$. However, many extensional system, such as FOIL [15], instead of checking that $q(e^-)$ is among the negative examples for $q$, just check that it is not among the positive. This is done because the set of negative examples, if specified, is probably not going to be complete and therefore we cannot expect it to contain all the negative examples. Instead, the set of positive example is considered to be, even if not complete, more representative of the concept because the number of positive examples in the universe is generally much smaller than that of negative ones. Therefore a closed-world assumption is made, assuming that all the non-positive atoms are negative. Negative examples are used only for testing the rules, not for extensional derivation. This can lead to intensional inconsistency, because when the definition of $q$ is learned, $q(e^-)$ may not be

among the negative examples and therefore we can derive a rule that covers it. This problem can be overcome by remembering the assumption made about $q(e^-)$, as it is done in the algorithm in [11], so that when we learn the rules for $q$, $q(e^-)$ is in the training set.

The second case in which this problem can arise is when the rule for $q$ is learned first

$$q(X) \leftarrow \ldots$$

The rule is tested against the examples of the training set and we are sure that this rule covers the positive examples and does not cover the negative ones. However, the training set is rarely equal to the entire universe, therefore some atoms can be undefined and some of these atoms can be as well covered by the generated rule. This can cause problems when we learn a rule for $p$

$$p(X) \leftarrow \ldots q(X)$$

because some of the undefined atoms $q(x^{\odot})$ intensionally covered by the rule for $q$, can result in the coverage of the negative example $p(x^{\odot})$ for $p$. There are two possible solutions to this problem:

- when the rule for $q$ is generated, all the undefined atoms are tested to determine if they are extensionally covered or not by the rule. If they are, they are added to the extensional definition of $q$. This is the approach followed in [11] and it is possible only when the Herbrand universe is finite, which is an usual assumption for extensional systems. However, even if the universe is finite, it can be very large, therefore this approach can be computationally expensive.

- Instead of testing all undefined atoms, we could test the rule for $p$ and when an atom is neither positive nor negative for $q$, we can assume it true in order to cover positive examples for $p$ or false in order to rule out negative ones. The assumptions made are recorded and the definition for $q$ is tested against them. If it is incomplete and/or inconsistent, it is revised accordingly. Otherwise, we can use the partial definition of $q$ when testing the rule for $p$ if there is no matching positive or negative example in the extensional definition of $q$, thus leading to an hybrid extensional-intensional system whose properties must be further investigated.

## 3.2 Extensional completeness, intensional incompleteness

This case happens when we learn a recursive program without a base clause. For example:

$even(X) \leftarrow succ(X, Y), odd(Y).$

$odd(X) \leftarrow succ(X, Y), even(Y).$

In this case, the problem can not be solved using abduction. A solution to this problem has been proposed in [13].

### 3.3 Intensional completeness, extensional incompleteness

This is the most obvious problem of extensionality, and the one that is best solved using ACL. Sometimes an intensionally complete hypothesis can not be learned because it is not extensionally complete, since some of the examples for $q$ needed to cover the positive examples for $p$ may not be available in the training set. Abduction can be used to assume all the needed examples for $q$ that are not known.

However, even if sometimes assumptions are the only way to overcome this problem, when other informations are available (such as a definition for $q$), they must be exploited as much as possible before resorting to abduction. We will show this in the next example adapted from example 4 in [16].

**Example 3.1** *Suppose you have to learn the concepts* $father$ *and* $male\_ancestor$ *from the training set*

$E^+ = \{father(luc, soetkin), father(bart, stijn),$
$father(willem, lieve),$
$male\_ancestor(luc, soetkin), male\_ancestor(bart, stijn),$
$male\_ancestor(rene, willem), male\_ancestor(rene, lieve)\}$
$E^- = \{father(lieve, soetkin), male\_ancestor(esther, lieve),$
$male\_ancestor(katleen, stijn)\}$

*and from the background knowledge*

$BK = \{parent(luc, soetkin), parent(bart, stijn),$
$parent(rene, willem), parent(willem, lieve),$
$male(luc), male(bart), male(rene), male(willem),$
$parent(lieve, soetkin), parent(esther, lieve), parent(katleen, stijn),$
$female(lieve), female(esther), female(katleen)\}$

*In this case, the following hypothesis is intensionally complete and consistent but it is extensionally incomplete because it does not cover* $male\_ancestor(rene, willem)$

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

*Suppose the definition of* $father$ *was learned first. In order to cover* $male\_ancestor(rene, willem)$, $father(rene, willem)$ *could be assumed but it could as well be derived using the definition learned for* $father$*. Using the definition learned so far is to be preferred to abduction when possible, because it leads to more certain results.*

The notion of extensionality should therefore be changed: when no definition is available for a predicate, use its training set for derivation, but when a definition has been learned, use that instead or in addition to the training set. What we get is the same hybrid system combining extensional and intensional coverage seen in section 3.1.

In the next example we show what kind of problem can be encountered when using assumptions to covers examples.

**Example 3.2** *Consider the same background knowledge and* $E^-$ *of the previous example and the training set*

$E^+ = \{father(luc, soetkin), father(willem, lieve)$
$male\_ancestor(luc, soetkin), male\_ancestor(bart, stijn),$
$male\_ancestor(rene, lieve)\}$

*In this case, the previous hypothesis is intensionally complete and consistent but is extensionally incomplete because it does not cover* $male\_ancestor(bart, stijn)$ *and* $male\_ancestor(rene, lieve)$

*The first two clauses could be learned as well by an extensional ACL system. When trying to cover* $male\_ancestor(bart, stijn)$*, the system could, correctly, abduce* $father(bart, stijin)$ *but it could do the same for* $male\_ancestor(rene, lieve)$ *obtaining* $father(rene, lieve)$*, which is not true in the intended interpretation, and the last rule would not be generated.*

*In order to discriminate between assumptions, we need some more information. This can be constituted by integrity constraints on the concept* $father$*, for example*

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

*or*

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

*which, respectively, state that "a father must be a parent" and "X can not be Y's father if it is Z's father and Z is Y's father". This kind of constraints can be given in the background knowledge or can be extracted as regularities from the data, however with the problems that will be shown in example 6.4.*

# 4 ACL for intensional systems

Extensional coverage has been introduced in order to solve the problems of intensional systems when used for Multiple Predicate Learning (MPL). These problems have been thoroughly analyzed in [16] and are summarized below.

Most intensional systems have been designed for single predicate learning. When we want to learn multiple predicates, the most straightforward way to solve the problem is to repeat the single predicate learning task for each target predicate. At each step, a decision has to be made on which predicate to learn next. The order in which predicates are learned is very important because for some orders we may not be able to find a solution even if one exists or we may find a very complex solution. Therefore, backtracking has to be considered on the predicate learning order. This problem is worsened by the fact that the restriction on the language imposed by the bias may further restrict the set of orders that lead to a solution. Moreover, in order to learn mutually recursive predicates, we have to interleave the generation of a clause for one predicate and

for the other, we can not learn the complete definition of a predicate after the other. Therefore, we must noy only consider all the orderings of single predicate learning tasks, but also all the orderings of single clause learning tasks.

This leads to a second problem: the addition of a consistent clause to a theory can make previous clauses inconsistent. The learning process is non-monotonic with respect to consistency. For example, suppose you learn 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. When we add the third clause, this can cover as well some of the atoms $q(e_p^-)$, because they may not be in $E_q^-$.

Therefore, in intensional systems, after the addition of a clause to the current partial hypothesis, the negative examples for all the predicates must be tested (global consistency), as in fact is done in the systems MPL [16] and TRACY [1]. This is clearly very expensive, and it is avoided by resorting to extensional derivation. This problem dos not arise in the case of Single Predicate Learning (SPL) because global and local consistency coincide, since all the negative examples are always tested in order to check consistency.

Among this two problems, the latter can be solved using abduction. When we test the clause

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

against a negative example $p(e_p^-)$ and the definition of $q$ is not complete (some $e_q^+$ are still in $E^+$), besides ensuring that $q(e_p^-)$ is not derivable, we should assume that $q(e_p^-)$ is false and add it to $E_q^-$, so that when a new clause for $q$ is learned these examples are not covered.

This can, however, lead to the impossibility of finding a solution even if it exists because the assumptions can constrain too much $q$. Therefore, backtracking on clause addition is still needed, besides backtracking on literals, and the assumptions made must be retracted. The cost of clause backtracking can be reduced using heuristics, both for the addition of literals and for the selection of the predicate in the head of the clause to learn. See [16] for an example of such heuristics.

Moreover, abduction can help considerably to learn more effectively in multiple predicate learning problems. In fact, in this case a possible incompleteness in the training set for a predicate can prevent the system to learn a correct definition for other predicates as well. By making assumptions about a subpredicate when the atom to be assumed cannot be derived by any rule, we can supply the incompleteness of the subpredicate definition. Then, the assumptions made can be used to learn new rules for the subpredicate, thus completing its definition. In other cases, the definition of the subpredicate may be incorrect because it is overgeneral, as will be illustrated in example 6.3.

### 4.1 ACL for learning normal logic programs

Normal logic programs are programs that can contain negated atoms in the body of the rules. When learning normal logic programs, apart from the previous two problems, another problem can arise: adding a clause to a partial hypothesis can reduce the coverage of that hypothesis. In fact, normal programs are non-monotonic, adding a clause to them may reduce the set of derivable facts. For example:

$q(X) \leftarrow Body_1(X).$
$p(X) \leftarrow Body_2(X), not\ q(X).$
$q(X) \leftarrow Body_3(X).$

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 SPL, 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 not anymore tested. This is the dual problem of the one seen before for definite programs. 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.

An alternative solution to this problem is to use abduction to record assumptions about negative literals $not\ p(e)$ being true (or $p(e)$ being false), so that clauses generated afterwards will not cover the example $p(e)$.

Therefore, to solve the problems of learning both definite and normal programs, we record negative assumptions about target predicates and add them to the set $E^-$ for successive learning phases. This avoids the checking of the whole training set, $E^+$ and $E^-$, after the addition of a new clause. In this approach, the set of negative examples is gradually expanded to include the negative assumptions made. This is similar to the approach followed in TRACY$^{not}$ [3]. Since we gradually add negative examples, this approach may seem similar to the one adopted in incremental systems such as MIS [18]. However, while in these systems a consistency check must be done after the addition of each $e^-$, we do not have to do this because we add an $e^-$ only after having tested that it is not covered by any clause.

## 5 Rules and integrity constraints

In this section, we examine in more detail the roles of rules and integrity constraints in learning. A target predicate may be defined by a set of rules with

that predicate in the head plus a set of integrity constraints with that predicate in the body.

The rules express sufficient condition for the target concept. This means that if an unseen example satisfies all the conditions in the body of a rule, we can classify it as an instance of the concept. If it does not satisfy all the conditions, we can consider it a negative instance of the concept. This corresponds to adopting a two-valued semantics.

Integrity constraints on target predicates, instead, represent necessary conditions that the unseen examples have to satisfy in order not to be classified as negative instances of the concept. In this way, an example is classified as negative if it does neither satisfy any sufficient condition nor all the necessary conditions. If an example does not satisfy any of the sufficient conditions but satisfies all the necessary ones, we cannot classify it either as positive or negative, but we have to consider it as unknown and we can make assumptions about it. This corresponds to adopting a three-valued semantics for the concepts, as observed by Flach [9].

A three valued semantics is necessary in order to learn in the presence of incomplete information, since we cannot consider everything that is not known to be false. In this case, we have to learn both the positive concept, represented by sufficient conditions, and the negative one, represented by necessary conditions, in order to distinguish between what is certainly true, what is certainly false and what is unknown. Therefore, rules and integrity constraints complement each other in the representation of the learned concept.

**Example 5.1** *Consider the abductive theory:*
$T = \{flies(X) \leftarrow bird(X), normal\_bird(X).\}$
$A = \{flies\}$.
$IC = \{ \leftarrow flies(X), penguin(X).\}$
*The rule in $T$ can be used to conclude that tweety flies if we know that it is a bird and is a normal_bird. If we know that freddie is a penguin, we can conclude that it does not flies. If we don't know anything about billy, we can assume that it flies because this is consistent with the integrity constraints.*

A different way in which rules and integrity constraints can complement each other is to use integrity constraints to specialize the rules. In the next section this case will be considered in detail.

# 6 Integrity constraints used to specialize rules

In top-down learners, rules are specialized by adding a literal to the body. Alternatively, integrity constraints can be used in order to specialize the rule. In this case, each fact derived by the rules must be checked against integrity constraints and, if it violates one or more constraints, it is rejected. Therefore, this use of constraints differs from the previous one in which they were used only to check

the assumptions made when no rule was applicable. Let us see an example of this use.

**Example 6.1** *Suppose the learning system has generated the rule:*
$flies(X) \leftarrow bird(X)$
*This rule is not correct because it can derive that penguins fly. We can specialize it by adding the constraint*
$\leftarrow flies(X), penguin(X).$

In the following, we will first examine the relationships between these two specialization operators, and then we will discuss what is the possible use of integrity constraints for specialization.

## 6.1 Equivalence of integrity constraints and addition of literals as specializing operators

Integrity constraints offer a way of specializing rules which is alternative to adding a literal to the body. We will show informally, by means of examples, that everything that can be expressed using integrity constraints in this way, can be expressed as well by adding literals to the rules. Let us start from the simplest case, in which we have a rule

$$c(X) \leftarrow Body(X).$$

where $c$ is the concept to be learned and $Body(X)$ is a conjunction of literals, and the constraint

$$\leftarrow c(X), p(X).$$

Informally, this theory expresses that "$c(X)$ is true if $Body(X)$ is true but we cannot have $c(X)$ and $p(X)$ true at the same time". The same theory can be expressed by the rule

$$c(X) \leftarrow Body(X), not\ p(X).$$

where we have used Negation As Failure (NAF) because the integrity constraints can be interpreted as meta-level statements about the provability of literals. For example, the previous constraint can be interpreted as

$$\leftarrow demo(T, c(X)), demo(T, p(X)$$

A more complex case is the one in which we have the constraint

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

This can be rewritten as

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

which, informally, can be interpreted as "it is never the case that $c(X)$, $p(X)$ and $not\ q(X)$ are true at the same time". Therefore, in order for $c(X)$ to be true, $p(X)$ must be false or $q(X)$ must be true. This can be expressed as well in terms of rules

$$c(X) \leftarrow Body(X), not\ p(X).$$

$$c(X) \leftarrow Body(X), q(X).$$

The last case we consider is the one in which some of the predicates in the integrity constraint are used to instantiate new variables not present in the concept atom

$$\leftarrow c(X), p(X,Y), q(X,Y)$$

In this case, we could not produce the previous two rules because we would loose the relationship between $X$ and $Y$. Therefore we have to keep the literals in both rules

$$c(X) \leftarrow Body(X), p(X,Y), not\ q(X,Y).$$

$$c(X) \leftarrow Body(X), not\ p(X,Y), q(X,Y).$$

These two rules do not cover the case in which both $p(X,Y)$ and $q(X,Y)$ are false, therefore we must add another rule

$$c(X) \leftarrow Body(X), not\ p(X,Y), not\ q(X,Y).$$

Alternatively, instead of the previous three rules, we could add the single rule

$$c(X) \leftarrow Body(X), not\ (p(X,Y), q(X,Y)).$$

Therefore, specialization by integrity constraints is not more expressive than specialization by addition of a literal.

## 6.2 Uses of integrity constraints as a specializing operator

We have seen that integrity constraints and the addition of literals are two alternative specialization operators. We want to investigate now when the use of integrity constraints is more appropriate.

In some cases, a correct rule can not be learned because of lack of training data, while it is possible to learn an integrity constraint, as it is illustrated in the next example.

**Example 6.2** *Suppose the learning system has generated the abductive theory:*
$T = \{father(X,Y) \leftarrow parent(X,Y).\}$
$IC = \{\leftarrow father(X,Y), female(X).\}$
*The first rule is not correct, because it can conclude that $X$ is a father even when $X$ is female. The integrity constraint is therefore used to avoid the derivation of female fathers. $not\ female(X)$ was not added to the rule because it was not useful to discriminate negative from positive examples, supposing we did not*

*have negative examples of female fathers. But analyzing the regularities in the data, it was possible to infer that it was never the case that a female was a father.*

In this case, the addition of a literal was not considered because of the way in which the rules are learned. We are supposing to use a top-down algorithm which learns the most general definition that is able to discriminate positive from negative examples. But in this way we may have overgeneralization, as shown in the previous example. A bottom-up learner would generate the most specific rule that covers all the positive examples and rules out the negative, but the opposite problem can arise: the generated rules could not generalize enough to cover unseen examples of the same concept. Moreover, in this case the most specific rule $father(X,Y) \leftarrow parent(X,Y), not\ female(X)$. contains a NAF literal, and bottom-up systems able to learn normal logic programs do not exist yet. Integrity constraints can therefore help to restrict the generality of the rules inferred by a top-down learner.

Moreover, we can see the integrity constraints as a means to generate negative informations which were not directly available. In the previous example, the constraints express that all female fathers are negative examples for the concept father, while in the training data we did not have any negative examples. In the following example we show how this negative informations can be fundamental for successive learning phases.

**Example 6.3** *Consider a multiple predicate learning task in which you have to learn the concepts father and grandfather starting from a background theory which contains the definitions for parent, male and female. Suppose also that the training set for father has the properties of example 6.2 and therefore we learn the same rule for father*

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

*Then we learn the definition for grandfather, obtaining*

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

*which is correct on the training set because all the positive example for grandfather are covered and none of the negative examples. The resulting theory, however, is clearly not correct on unseen cases, in order to make it correct, we have to add the constraint*

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

One objection can be made regarding example 6.2: a correct rule could not be learned because the available data was not sufficient. In this case the generation of integrity constraints resulted in the correct definition of the concept, but maybe in other cases it could find regularities in the data that are irrelevant to the concept that we are trying to learn.

**Example 6.4** *Suppose you have the same learning problem of example 6.2 where now you know a person steve that is not a father and that is poor*

*(poor(steve)). If you do not know anything about fathers being poor or not, you could assume that they are not poor and observe the regularity in the data*
$\leftarrow father(X,Y), poor(X)$.
*which expresses that there are no poor fathers. This is derived in the same way as it was the constraint on female, but it restricts too much the concept father*

This example shows that we must be very careful when extracting regularities from data because sometimes incidental regularities could be found. We need therefore a way to distinguish between incidental from relevant regularities, which must come from other information than the training data, such as the user bias. In example 6.2, we could generate only the constraint $\leftarrow father(X,Y), female(X)$ because the user somehow knows that the concept father has something to do with sex but not with welfare of people.

This problem is part of a wider issue regarding the minimum amount of data which is necessary for learning a theory. With ACL we are trying to lower this limit by making assumptions, but the available data must guide these assumptions. If the data are insufficient, we can make the wrong assumptions and do not learn a correct definition for the target concepts. In this paper we assume that the amount of information available is sufficient for learning the intended theory using ACL. Determining what is exactly this amount is an open problem that is subject for future works.

In the following, we assume that integrity constraints are not used to specialize rules but only to restrict the possible assumptions when no rule is applicable.

# 7 Algorithm

In order to develop an algorithm able to perform ACL in its full form, we can start from the intensional algorithm presented in [8] in which the usual notion of coverage of examples is substituted with the notion of abductive coverage. It is able to make assumptions during the learning phase and to generate rules that contain abducibles.

We want to develop an intensional system because we have shown that the problems of intensionality can be solved with abduction. Therefore we consider the algorithm in [8] and we extend it in order to perform MPL, using ideas from [16], and to learn general integrity constraints of the form

$$A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$$

where $A_i$ and $B_i$ are atoms. The algorithm considered is also able to learn integrity constraints, but only of the limited form of denials between a predicate and its default literal

$$\leftarrow a(X), not\_a(X)$$

General integrity constraints can be learned by ICL [17]. ICL starts from a definite program as background knowledge, from a set of positive and a set of

```
procedure PACL(
    inputs : E+, E− : training sets,
        AT = ⟨T, A, IC⟩ : background abductive theory,
    outputs : H : learned theory, Δ : abduced literals)

H := ∅
Δ := ∅
while E+ ≠ ∅ do
    GenerateRule(input: AT, H, E+, E−, Δ; output: Rule)
    if GenerateRule fails, then
        do backtracking on the previous rule added
    Let E+_Rule be the set of positive examples covered by Rule
    Let Δ_Rule be the set of literals abduced during
        the derivation of e+ and not e− using 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
    H := H ∪ {Rule}
    Δ := Δ ∪ Δ_Rule
endwhile
output H, Δ
```

Figure 1: PACL, the covering loop

negative interpretations and finds a clausal theory $H$ such that all the positive interpretions (together with the background knowledge) are models of $H$ and all the negative ones are not models. We combine ICL with an extension of the previous algorithm in order to perform ACL in its full form.

In section 7.1 we will discuss how to extend the algorithm in [8] in order to perform MPL using best first search. We will call it PACL (Partial Abductive Concept Learning). In section 7.2 we will show how to use ICL in our framework, and in section 7.3 we describe how to integrate the two in order to obtain ACL, an algorithm that performs full ACL.

## 7.1 Partial ACL

This algorithm is an extension of the algorithm presented in [8] which in turn is based on the generic top-down algorithm presented in [12]. As the generic top-down algorithm, PACL is composed of two loops: the covering loop (figure 1) and the specialization loop (figure 2).

The covering loop starts with an empty hypothesis and, at each iteration,

**procedure** GenerateRule(
    **inputs :** $AT$ : background theory, $H$ : current hypothesis,
        $E^+, E^-$ : training sets, $\Delta$ : current set of abduced literals
    **outputs :** $Rule$ : rule)

$StartRules := \{\ p(X) \leftarrow true.$ where $p$ is a target predicate for which
    there are still some $e^+$ in $E^+\ \}$
$Agenda := \emptyset$
**for all** $Rule \in StartRules$ **do**
    add $\langle Rule$,Evaluate$(Rule, AT, E^+, E^-, \Delta), \rangle$ to $Agenda$
Select and remove $Best$ rule from $Agenda$
**While** $Best$ covers some $e^-$ **do**
    $BestRefinements :=$ set of refinements of $Best$ allowed
        by the language bias
    **for all** $Rule \in BestRefinements$ **do**
        $Value :=$ Evaluate$(Rule, AT, E^+, E^-, \Delta)$
        **if** $Rule$ covers at least one pos. ex. **then**
            add $\langle Rule, Value \rangle$ to $Agenda$
    **endfor**
    Select and remove $Best$ rule from $Agenda$
**endwhile**
let $Best$ be $\langle Rule, Value \rangle$
output $Rule$

Figure 2: PACL, the specialization loop

**function** Evaluate(
**inputs :** Rule: rule ,$AT$ : background theory,
$E^+, E^-$ : training sets, $\Delta$ : current set of abduced literals)
**returns** the value of the heuristic function for $Rule$

$n^{\oplus} = 0$, number of pos. ex. covered without abducing anything
$n^{\oplus}_A = 0$, number of pos. ex. covered abducing something
$n^{\ominus} = |E^-|$, number of neg. ex. covered ($not\ e^-$ has failed)
$n^{\ominus}_A = 0$, number of neg. ex. uncovered abducing something
$\Delta_{in} = \Delta$
**for each** $e^+ \in E^+$ **do**
**if** AbductiveDerivation$(e^+, \langle T \cup H \cup \{Rule\}, A, IC\rangle, \Delta_{in}, \Delta_{out})$
succeeds **then**
**if** no new assumptions have been made **then**
increment $n^{\oplus}$
**else**
increment $n^{\oplus}_A$
**endif**
**endif**
$\Delta_{in} = \Delta_{out}$
**endfor**
**for each** $e^- \in E^-$ **do**
**if** AbductiveDerivation$(not\ e^-, \langle T \cup H \cup \{Rule\}, A, IC\rangle, \Delta_{in}, \Delta_{out})$
succeeds **then**
decrement $n^{\ominus}$
**if** new assumptions have been made **then**
increment $n^{\ominus}_A$
**endif**
**endif**
$\Delta_{in} = \Delta_{out}$
**endfor**
return Heuristic$(n^{\oplus}, n^{\oplus}_A, n^{\ominus}, n^{\ominus}_A)$

Figure 3: PACL, evaluation of the heuristic function

a new clause is added to the hypothesis. Each clause is generated in the specialization loop (contained in the function GenerateRule). The new clause will cover some positive examples and none of the negative, after the addition the covered positive examples are removed from the set of positive examples $E^+$ and the literals abduced during the derivation of positive and negative examples are added to the current set of abducibles $\Delta$. Moreover, each assumption made about target predicates is added either to $E^+$ or to $E^-$, depending on the sign of the literal. The loop ends when all the positive examples are covered.

The space of possible clause orderings is searched depth-first: in the case in which no clause could be generated by GenerateRule, then backtracking is done on the clause added at the previous step. The clause is retracted and the procedure GenerateRule is requested to produce another clause.

Let us consider the specialization loop. The clauses are generated by performing a best-first search in the space of the possible refinements. At the beginning, the set of current clauses (called *Agenda*) contains one clause with an empty body for each target literal whose definition has not yet been completed (some positive examples for it are still in $E^+$). For each clause in the set the value of the heuristic function is computed (figure 3). This is done by trying to derive each positive example and the negation of each negative one.

At each iteration of the specialization loop, the clause with the maximum value for the heuristic function is selected and removed from the *Agenda*. All its refinements, allowed by the language bias, are generated. For each refinement, the value of the heuristic function is calculated and the refinement, together with the value of the heuristic, is added to the *Agenda* if it covers at least one positive example. The loop terminates when a clause is found that does not cover any negative examples. That clause is then returned as output. In the case in which such a clause could not be found, the procedure fails. This happens when the *Agenda* becomes empty, because no refinement covers at least one positive example.

Let us now consider the way in which the heuristic function is evaluated. For each positive example $e^+$ and for each negative example $e^-$, an abductive derivation is started for $e^+$ and $not\ e^-$ respectively, using the procedure defined in [10]. This procedure takes as input the goal to be derived, the abductive theory and, if it succeeds, produces as output the set of literals abduced during the derivation. We consider as well as input the set of literals already abduced. This does not require any major modification of the procedure. In this way, we ensure that the assumptions made during the derivation of the current example are consistent with the ones made before. Thus, we can test each positive and negative example separately and be sure that the clause will derive $e_1^+ \wedge \ldots \wedge e_n^+ \wedge not\ e_1^- \wedge \ldots \wedge not\ e_m^-$. The set of abduced literals is passed on from derivation to derivation and gradually extended. This is done as well across different clauses, in order to maintain the consistency among assumptions.

The positive and negative examples covered are counted, distinguishing between examples covered (uncovered) with or without abduction, and these num-

bers are used to calculate the heuristic function, which will be discussed in section 8.

However, it must be noted that an example could be successfully derived with different sets of assumptions. Suppose you have two possible sets, one compatible with some examples while the other not compatible. If we assume the second set, we are not going to derive anymore these examples. This is a choice point of the algorithm and backtracking should be used in the case of failure of the generation of a clause in order to find other set of abducible that could possibly allow the coverage of more examples. However, we decided not to consider this choice point in order to reduce the computational cost of the algorithm.

The strategy for generating the clause to add to the partial theory is similar to one used in the system MPL because the predicate in the head of the clause is not decided a priori, before starting the specialization, but is selected in order to produce the most promising clause. However our strategy differs from MPL because they keep a set of the best bodies found so far while we keep the set of the best complete clauses found so far. They evaluate one body by selecting the maximum value of the heuristic function for the clauses generated by adding all the possible heads to the body. We think that in most cases each body in the set will have a good performance for just one head and therefore it is useless to test it with all the other heads.

Let us examine the properties of soundness and completeness the algorithm. The use of literals abduced so far in the derivation of the examples, ensures that we will not generate clauses that make previous clauses inconsistent or that reduce their coverage. Therefore the algorithm is sound for the problem of ACL and, as a consequence, it is sound also for learning normal logic programs in the case of multiple target predicates. As regards completeness, depth-first search in the space of clause orderings and best-first search in the space of clause refinements ensure a complete search of the space in the worst case. However, as observed before, we do not perform backtracking on the choice of the abduced literal set. Therefore the algorithm is not complete for the problem of ACL. If backtracking on that choice point is added, then the algorithm becomes complete for ACL and therefore also for learning normal logic programs with multiple target predictes.

The space of clause refinements can be represented as a tree, in which the root is the empty clause and each child of the root is a clause with the target predicate in the head and an empty body. The complete search space can therefore be represented by a tree in which each node is also a tree. In order to be able to do backtracking on the clauses added, we need to store the boundary nodes of the inner tree for each iteration of the covering loop (i.e., for each node of the outer tree). This means storing the *Agenda* of the specialization loop. In this way, when backtracking is required, the previous *Agenda* is restored and the procedure GenerateRule selects from it another clause which does not cover negative examples, maybe after having done some specialization steps. Since

the search strategy of the outer tree is depth first, only the current path along the tree must be stored and therefore we need to store one set of clauses for each level of the tree. This may be too expensive from a storage point of view and we could use beam-search instead of best-first, thus putting a constraint on the size of the $Agenda$. However, this restrict further the property of completeness of the algorithm.

A number of optimizations are possible. One regards the search space: each time a new clause is added, the search in the space of refinements starts from the root of the tree and goes down until it finds a clause which is consistent, i.e. does not cover any negative examples. When we are looking for the next clause, we do not need to start from the root because we already know that all the clauses up to the point reached before were inconsistent at the previous step, and therefore will be inconsistent as well now, since $E^-$ and $\Delta$ have been, possibly, enlarged. Therefore, we could start from the set of clauses obtained at the previous step. Clearly, the value of the heuristic function must be recalculated for each clause in the set because the sets $E^+$, $E^-$ and $\Delta$ may have changed. This optimization saves a considerable amount of search time and does not require additional storage space because the set of clauses of the previous step must be kept anyway in order to be able to perform backtracking on the outer tree.

Another optimization regards the evaluation of the heuristic function for each refinement of a clause. This evaluation requires to try an abductive derivation for each member of $E^+$ and $E^-$. As a result, we identify the subsets $E^+_{Ref} \subseteq E^+$ and $E^-_{Ref} \subseteq E^-$ that contain positive and negative examples still covered by the refinement. At the successive specialization step for this clause, we do not need to test all the elements of $E^+$ and $E^-$ because the ones which were not covered before will not be covered as well by a refinement (since we are starting from the same set of abduced literals). Therefore, we can test only the subsets $E^+_{Ref}$ and $E^-_{Ref}$, thus gradually reducing the number of tests as the clause is specialized. This requires storing, together with the refinement and its value, also the subsets $E^+_{Ref}$ and $E^-_{Ref}$, together with the set $\Delta_{Ref}$ of literals abduced during previous tests, so that the new tests will be consistent with them.

## 7.2 Learning integrity constraints

In order to learn the integrity constraints we use the system ICL [17]. The learning problem that is solved by ICL can be stated as follows.

- **Given**
  - $B$, a definite clause background theory;
  - $P$, a set of interpretations such that for all $p \in P$, $M(B \cup p)$ is a true interpretation of the unknown target theory;

- $N$, a set of interpretations such that for all $n \in N$, $M(B \cup n)$ is a false interpretation of the unknown target theory.

- **Find** a clausal theory $H$ such that
  - for all $p \in P$, $M(B \cup p)$ is a true interpretation of $H$ (Completeness);
  - for all $n \in P$, $M(B \cup n)$ is a false interpretation of $H$ (Consistency);

In our case, we would like to learn integrity constraints on abducibles by using the information contained in the set $\Delta$ of assumption made during the learning phase. $\Delta$ is a set of positive and negative literals, we can obtain from it two sets: $\Delta^+$, which contains all the positive literals, and $\Delta^-$, which contains the atoms corresponding to negative literals in $\Delta$. In the following, $E_A^-$ indicates the set of negative examples of abducible predicates. We can apply ICL to our case in this way:

- the background knowledge $B$ is the program $T'$,
- the set $P$ contains only one positive interpretation $p = \Delta^+$;
- the set $N$ contains a negative interpretation for each element of $\Delta^- \cup E_A^-$

The training sets are included in the interpretations if the target predicates are also abducible, and therefore we would like to learn integrity constraints on them. The learning bias for ICL is set in order to learn clauses of the form

$$head \leftarrow body, abducible\_predicate$$

The learned clauses will be true in $M(T' \cup \Delta^+)$ and will be false in each model $M(T \cup \{\delta^-\})$ where $\delta^- \in \Delta^- \cup E_A^-$. Therefore, when the integrity constraints will be added to the final abductive theory, they will not allow any assumption belonging to $\Delta^- \cup E_A^-$, that is precisely what we want.

In our case, $T$ is a normal program, while in the problem statement of ICL the background knowledge is considered to be a definite program. However, even if the authors did not mention it in [17], ICL can be used without modifications also for learning from normal background knowledge.

## 7.3 Full ACL

A system for performing full ACL can be obtained by running in sequence PACL and ICL:

- Input: $\langle T, A, IC \rangle, E^+, E^-$
- Run PACL obtaining $H, \Delta$
- Run ICL, obtaining IC', with input:

- $B = T$
- $P = \{\Delta^+\}$
- $N = \{\{\delta^-\}$ such that $\delta^- \in E_A^- \cup \Delta^-\}$

- Output: $\langle T \cup H, A, IC \cup IC' \rangle$

# 8 Heuristics

Various heuristics have been proposed in ILP for the evaluation of the quality of a clause, see [12] for an overview. All of them express the intuitive notion that a clause is better than another if it covers more positive and less negative examples. The most widely used heuristics can be divided into two main families, one based on *expected classification accuracy* and the other based on *informativity*. Both of them are based on the probability $p(\oplus|c)$ that an example covered by a clause $c$ is positive. However, this probability is not known and therefore must be estimated. After having presented the various heuristics as functions of $p(\oplus|c)$, we will discuss how to estimate $p(\oplus|c)$. Then we will show how the heuristics presented can be adapted to the ACL framework.

## 8.1 Heuristic functions

The *expected classification accuracy* of a clause $c$ is defined exactely as the probability that an example covered by $c$ is positive

$$A(c) = p(\oplus|c)$$

The *informativity* of a clause $c$ is defined as the amount of information necessary to specify that an example covered by the clause is positive

$$I(c) = -\log_2 p(\oplus|c)$$

In this case, as opposed to the previous one, the quality of the clause is higher if informativity is lower. A more intuitive meaning of informativity will be given in section 8.2, when the probability estimates will be discussed.

From these two basic heuristics, others can be derived. Given the current clause $c = T \leftarrow Q$ and a refinement $c' = T \leftarrow Q'$, the *accuracy gain* $AG(c', c)$ and *information gain* $IG(c', c)$ can be defined as follows

$$AG(c', c) = A(c') - A(c) = p(\oplus|c') - p(\oplus|c)$$

$$IG(c', c) = I(c) - I(c') = \log_2 p(\oplus|c') - \log_2 p(\oplus|c)$$

The accuracy gain is the increase in classification accuracy, while the information gain is the decrease in information necessary to specify that an example covered by the clause is positive, achieved by specializing clause $c$ to $c'$. In this way we

can compare different specializations of a clause and select the one which gives the biggest gain.

However, these heuristics can prefer very specific rules with an high gain to more general rules whith a lower gain. For example, two rules could be obtained from the specialization of the same rule, one which covers 2 positive examples and no negative ones, the other which covers 100 positive examples and 1 negative. The gain heuristics would prefer the first rule, even if the second is probably more interesting because it is more general. Therefore, the value of the gain heuristics is adjusted by means of a weight factor which should take into account the generality of the rule. This factor can be given by the fraction of positive examples which is covered before and after the specialization step $\frac{n^{\oplus}(c')}{n^{\oplus}(c)}$. What we get is *weighted accuracy gain* $WAG(c', c)$ and *weighted information gain* $WIG(c', c)$

$$WAG(c', c) = \frac{n^{\oplus}(c')}{n^{\oplus}(c)} \times (p(\oplus|c') - p(\oplus|c))$$

$$WIG(c', c) = \frac{n^{\oplus}(c')}{n^{\oplus}(c)} \times (\log_2 p(\oplus|c') - \log_2 p(\oplus|c))$$

By introducing a weight, we aim at finding a balance between the gain and the number of positive examples covered by a clause. An heuristic similar to $WIG(c', c)$ is used in FOIL [15].

It has been shown that the respective heuristics based on accuracy and informativity give similar results with respect to the accuracy of the generated clauses [5]. Instead, the model used to estimate probability has a stronger influence on the performances of the heuristic, specially when the training data are noisy.

## 8.2 Probability estimates

All the heuristic functions previously described are based on the probability $p(\oplus|c)$ that an example covered by clause $c$ is positive. Clearly, this probability can not be known with certainty, because it is given by the behaviour of the clause on all the examples, while we have only a small sample constituted by the training set. Therefore, we must estimate $p(\oplus|c)$ using only the examples in the current training set $E_{cur}$ and, in particular, the numers:

- $n^{\oplus}(c)$ = number of positive examples covered by $c$
- $n(c)$ = total number of examples covered by $c$

Three different estimates have been proposed in the literature: *relative frequency, Laplace estimate* and *m-estimate*, in order from the most accurate to

the least. The simplest of them is the relative frequency of positive examples with respect to the total number of examples

$$p(\oplus|c) = \frac{n^{\oplus}(c)}{n(c)}$$

By using this probability estimate in the informativity heuristic, we obtain

$$I(c) = \log_2 n(c) - \log_2 n^{\oplus}(c)$$

This formulae can be used to give a more intuitive meaning of the informativity heuristic. Suppose we have $n(c)$ examples and we are testing them. Suppose that we want to transmit a message expressing the fact that the example tested is positive. The informativity of the clause is the number of bits necesary to encode this message and it is given by the number of bits necessary to specify which example was tested, $\log_2 n(c)$, minus the number of bits necessary to specify which of the positive examples was covered, $\log_2 n^{\oplus}(c)$, because we are not interested in "which" positive example was but only that it was a positive example.

This probability estimate is the simplest and the most diffused. However, the reliability of this estimate decreases as the size of the training set decreases: in the extreme case of only one positive example in $E_{cur}$, the estimate of $p(\oplus|c)$ is 1. This is clearly an estimate too optimistic even in the absence of noise. To avoid this problem, the Laplace law of succession was used [14]: if in the sample of $n$ trials there were $s$ successes, the probability of the next trial being successful is $\frac{s+1}{n+2}$, assuming a uniform initial distribution of successes and failures. The Laplace estimate is therefore given by

$$p(\oplus|c) = \frac{n^{\oplus}(c) + 1}{n(c) + 2}$$

This estimate is more reliable when dealing with a small number of samples. For example, in the case in which both $n^{\oplus}(c)$ and $n(c)$ are 0, the probability is $\frac{1}{2}$, which reflects the fact that an empty training set can not alter our a priori assumptions that positive and negative examples have the same probability.

However, this assumption is rarely true in practice. Therefore the $m$-estimate [4] was introduced that takes into account as well the prior probabilities of the classes

$$p(\oplus|c) = \frac{n^{\oplus}(c) + m \times p_a(\oplus)}{n(c) + m}$$

where the prior probability $p_a(\oplus)$ can be estimated by the relative frequency of positive examples in the initial training set $\frac{n^{\oplus}}{n}$. The value of $m$ expresses our confidence in the representativeness of the training set. The actual value of $m$ should be set subjectively according to the amount of noise in the examples (larger $m$ for more noise). As $m$ grows towards infinity, the $m$-estimate approaches the prior probability of the positive class.

The $m$-estimate provides a stronger theoretical ground to heuristc functions and allows to build clauses that are more accurate on unseen examples. Moreover, the other two probability estimates can be obtained as special cases of the $m$-estimate by appropriately setting the two parameter $m$ and $p_a(\oplus)$:

- relative frequency $p(\oplus|c) = \frac{n^{\oplus}(c)}{n(c)}$ for $m = 0$, and
- Laplace estimate $p(\oplus|c) = \frac{n^{\oplus}(c)+1}{n(c)+2}$ for $m = 2$ and $p_a(\oplus) = \frac{1}{2}$.

In [12] the authors report the results of experiments conducted in order to compare the performances of $AG_{rf}$, $AG_{Lap}$, $AG_{m=2}$, $WAG_{rf}$, $IG_{rf}$, $IG_{Lap}$, $IG_{m=2}$, $WIG_{rf}$, obtained by combining the accuracy gain and information gain with the different probability estimates and weighted gains with relative frequency. The results of the experiments showed that $WAG$ and $WIG$ have performance similar to unweighted $AG$ and $IG$ using the $m$-estimate. Therefore the authors conclude by making the hypothesis that weighted heuristics perform better than unweighted ones, the best being $WIG$ and $WAG$ using the $m$- estimate.

## 8.3 A heuristic for ACL

We have seen that informativity and accuracy have similar performances. Therefore we have chosen accuracy for its simplicity. The gain heuristics give an estimate of the quality improvement obtained by specializing a a clause. In our case we are more interested in selecting the clause which has the best performance on the training set, not the one which has had the best improvement. Therefore we use simple accuracy as the heuristic function.

As regards the function used to estimate probability, we use $m$-estimate because it has been shown to have better performances. We now illustrate how to apply this probability estimate in the abductive framework. For the sake of simplicity, in the following discussion we will use relative frequency. The modifications to this estimate can then be extended to the $m$-estimate without difficulties.

In ACL a new clause $c$ is tested against the examples of the training set by starting an abductive derivation for each positive example and for the negation of each negative one. Some positive examples will be covered without abducing any literals, while other examples will be covered by making some assumptions. Similarly, some negative examples will be ruled out abducing some facts, while others without abducing anything. As a consequence, in this case we have four figures to consider

- $n^{\oplus}(c)$ = number of positive examples covered by $c$ without abduction of any literal
- $n^{\ominus}(c)$ = number of negative examples covered by $c$ ($not\ e^-$ has failed)

- $n^{\oplus}_{Abd}(c)$ = number of positive examples covered by $c$ with the abduction of some literals
- $n^{\ominus}_{Abd}(c)$ = number of negative examples uncovered by $c$ (*not* $e^-$ has succeded) with the abduction of some literals

The examples covered by making some assumptions are not covered with complete certainty, because abduction is a form of hypothetical reasoning. The assumptions are hypothesis that may turn out to be wrong, the integrity constraints ensure that we do not make an assumption when it is certainly false, but if no constraint is violated the hypothesis may still be false, because we may not have all the relevant integrity constraints. Therefore, we should give less importance to examples covered or ruled out by abducing some literals. In other words, if two clauses cover the same amount of positive and negative examples but one of them covers the positive examples by making assumptions, than we should prefer the other. In the case in which they both cover some examples with abduction and some other without abduction, then a balance should be found. The heuristic function we propose is

$$A = \frac{n^{\oplus} + k \times n^{\oplus}_{Abd}}{n^{\oplus} + n^{\ominus} + n^{\oplus}_{Abd} + (1-k) \times n^{\ominus}_{Abd}}$$

where we have omitted for brevity the argument $c$ from the figures. The coefficient $k$, with $0 \leq k \leq 1$, expresses the confidence we have in the assumptions that can be derived from the background abductive theory. In the numerator, positive examples covered with the abduction of some literals have been given a smaller weight, expressed by $k$, with respect to the ones covered without abduction. It may seem natural to add the same weight to $n^{\oplus}_{Abd}$ as well in the denominator. This has not been done because of the following observation. Consider two rules for which $n^{\ominus}_{Abd}$ and $n^{\ominus}$ are both 0. If we have the $k$ factor also at the denominator, the value of the heuristic function will be 1 for both of the rules regardless of the fraction of positive examples that are covered with abduction $\frac{n^{\oplus}_{Abd}}{n^{\oplus}}$, while we would prefer the rule that covers more positive examples without abduction.

The negative examples ruled out by using abduction are taken into account with the addendum $(1-k) \times n^{\ominus}_{Abd}$ in the denominator. The reason for this can be explained as follows. Consider a rule which would cover $n'^{\ominus}$ negative examples if abduction were not available, but using abduction we can rule out $n^{\ominus}_{Abd}$ negative examples. Therefore the covered negative examples are $n^{\ominus} = n'^{\ominus} - n^{\ominus}_{Abd}$. If we apply the accuracy heuristic of standard ILP to the second rule we get

$$A = \frac{n^{\oplus}}{n^{\oplus} + n'^{\ominus} - n^{\ominus}_{Abd}}$$

Subtracting $n^{\ominus}_{Abd}$ in the denominator influences too strongly the heuristic function because a negative example ruled with abduction is considered equivalent

to a negative example ruled out without abduction. Therefore we multiply $n^{\ominus}_{Abd}$ by $k$ in order to reduce its influence, obtaining

$$A = \frac{n^{\oplus}}{n^{\oplus} + n'^{\ominus} - k \times n^{\ominus}_{Abd}}$$

We know that $n'^{\ominus} = n^{\ominus} + n^{\ominus}_{Abd}$, therefore we have

$$\begin{aligned} A &= \frac{n^{\oplus}}{n^{\oplus} + n^{\ominus} + n^{\ominus}_{Abd} - k \times n^{\ominus}_{Abd}} \\ &= \frac{n^{\oplus}}{n^{\oplus} + n^{\ominus} + (1-k) \times n^{\ominus}_{Abd}} \end{aligned}$$

# 9 Extension of the algorithm to ACL1

The algorithm we have defined complies with the definition of ACL2, in which it is required that the negation of each negative example is abductively entailed by the learned theory. We want now to adapt the algorithm to comply with the definition of ACL1, in which it is required that the learned theory must not abductively entail each negative example. The condition of ACL1 is more stringent than that of ACL2, because in ACL2 we can make assumptions in order to rule out a negative example, while in ACL1 not only we can not make any assumption to rule the example out, but it is required also that no assumption can be made in order to cover it.

Therefore, the algorithm must be changed so that, instead of testing the success of the abductive derivation of $not\ e^-$, it tests the failure of the abductive derivation of $e^-$. The heuristic function must be changed accordingly: in this case we do not to distinguish between negative examples ruled out with abduction or without abduction, therefore the heuristic can be simplified in

$$A = \frac{n^{\oplus} + k \times n^{\oplus}_{Abd}}{n^{\oplus} + n^{\ominus} + n^{\oplus}_{Abd}}$$

The condition on negative examples of ACL1 can be too stringent in some cases, thus we may look for an hybrid solution: if a negative example is still covered according to ACL1, we may try to rule it out using abduction as specified in ACL2. However, a negative example not covered according to ACL1 and one not covered according to ACL2 can not be considered equivalent in the evaluation of the clause, because the latter fact is less certain than the first. The set of negative examples is partitioned into three classes:

1. examples not covered by the clause ( $\leftarrow e^-$ failed)
2. examples ruled out with abduction ( $\leftarrow e^-$ and $\leftarrow not\ e^-$ both succeeded[1] )

[1] with different sets of assumptions.

3. examples covered by the clause ( $\leftarrow e^-$ succeeded and $\leftarrow not\ e^-$ failed)

If we indicate the cardinality of the last two classes with the names $n^{\ominus}_{Abd}$ and $n^{\ominus}$, we can use the same heuristic function seen in section 8.3. However, in this case the classes are different from those considered before

1. examples not covered by the clause ( $\leftarrow not\ e^-$ succeeded with $\Delta = \emptyset$)
2. examples ruled out with abduction ( $\leftarrow not\ e^-$ succeeded with $\Delta \neq \emptyset$)
3. examples covered by the clause ( $\leftarrow not\ e^-$ has failed)

Since the two partitions are different, the value of $k$ would have to be different in the two cases in order to reflect the different confidence we have in the uncoverage of negative examples.

# 10 Conclusions

In this paper we have investigated how abduction and induction can be integrated in order to obtain a more powerful learning framework in the context of ILP. We have adopted the Abductive Concept Learning framework defined in [6] which extends the ILP learning paradigm to the case in which both the background and the target theory are abductive logic programs. In this framework, we have shown how we can learn in the presence of incomplete information in the background knowledge and/or in the training set by exploiting the hypothetical reasoning of abduction.

After having stated the requirements for an algorithm that performs ACL, we illustrated the benefits of applying such an algorithm to extensional and intensional ILP systems. As regards extensional systems, the problems of extensional consistency, intesional inconsistency and intensional completeness, extensional incompleteness can be solved by using abduction and recording the assumptions made. As regards intensional systems, abduction can be used to avoid the checking of the global consistency of the current hypothesis after the addition of each clause, in the case of multiple predicate learning. Moreover, when learning normal logic programs, abduction can be used in order to avoid the rechecking of positive examples previously covered after the addition of a clause.

Abductive logic programs are a powerful means of knowledge representation. They allow the use of integrity constraints in order to represent concepts. We have investigated two different uses of integrity constraints, the first consist in having correct rules and using constraints only to check assumptions when no rule is applicable, while the second consist in having overgeneral rules which are specialized by using the constraints. We have shown some of the problems connected with the latter use and therefore we have adopted the former.

Finally, we have proposed an algorithm for ACL, which performs a depth-first search in the space of clause orderings and a best-first search in the space of

clause refinements, together with an appropriate heuristic function. This algorithm is sound but it is not complete because it does not consider backtracking on abductive explanations of examples. If backtracking is added for this choice point, the algorithm will also be complete. However, we think that adding this backtracking point would be computationally too expensive in practice.

In the future, the algorithm that has been presented will be implemented and a number of experiments will be performed in order to evaluate the advantages of the ACL system with respect to conventional ILP systems such as FOIL [15] or m-FOIL [7]. In order to show the performances of ACL on incomplete data, we will consider a complete dataset and we will take gradually information out, then comparing the degradation of performances of ACL with those of FOIL or m-FOIL. Moreover, we will consider the application of ACL to a domain where the background knowledge is inherently abductive and therefore conventional ILP systems can not be applied. The domain is event-calculus and the learning problem consist in finding rules which describe the effect of actions given a sparse training set containing facts about actions and properties holding at different time instants.

According to the results of these experiments, the parameters in the design of the algorithm will be tuned in order to get the best ratio of quality of learned program to learning time. For example, if backtracking on clause orderings proves to be too expensive, a greedy search will be performed instead. At the same time, if the best-first search strategy proves to be too expensive in terms of time or space, we could resort to beam-search, thus putting a limit on the size of the agenda, and eventually to hill-climbing, by setting to 1 that size. Moreover, the parameters $k$ and $m$ in the heuristic function have to be set and different experiments will be performed in order to evaluate the effect of these parameters on the learning process.

Further work is required also on theoretical aspects. The different uses of integrity constraints, as specializng operator or as condition only on assumptions, must be further investigated. By combining the learning of rules with that of integrity constraints, ACL integrates the two learning paradigm of explanatory and confirmatory induction [9]. The relation between these two paradigms in the context of ACL must be better understood, in order to have a clearer idea of the capabilities and the limits of the framework. Another interesting research direction is to deepen the study of learning in a three-valued setting. In particular, in such a setting we should learn as well the definition of the negation of concepts. Rules, being sufficient conditions, can express positive concepts, while integrity constraints, being necessary conditions, can express negative concepts, in the sense that if a constraint is violated by the assumption of a fact, that fact is certainly false. In this case, when an unseen example has to be classified, first we may try to classify it with certainty as a positive or negative instance of a concept and, only if it is not possible, we can make an hypothesis about it.

# References


[1] F. Bergadano and D. Gunetti. Learning Clauses by Tracing Derivations. In *Proceedings 4th Int. Workshop on Inductive Logic Programming*, 1994.

[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] B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the Ninth European Conference on Artificial Intelligence*, pages 147–149, London, 1990. Pitman.

[5] B. Cestnik. *Estimating probabilities in machine learning.* PhD thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia, 1991. In Slovenian.

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

[7] S. Džeroski and I. Bratko. Handling noise in inductive logic programming. In S. Muggleton, editor, *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, Report ICOT TM-1182, 1992.

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

[9] P. Flach. *An inquiry concerning the logic of induction.* PhD thesis, Tilburg University, 1995.

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

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

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

[13] L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In *Advances in Inductive Logic Programming.* IOS Press, 1996.

[14] T. Niblett and I. Bratko. Learning decision rules in noisy domains. In M. Bramer, editor, *Research and Development in Expert Systems III*, pages 24–25. Cambridge University Press, 1986.

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

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

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

[18] E. Shapiro. *Algorithmic Program Debugging.* MIT Press, 1983.