Learning Non-Monotonic Logic Programs: Learning Exceptions

Yannis Dimopoulos and Antonis Kakas
Department of Computer Science
University of Cyprus
CY-1678, Nicosia, Cyprus
{yannis, antonis}@turing.cs.ucy.ac.cy

Abstract. In this paper we present a framework for learning non-monotonic logic programs. The method is parametric on a classical learning algorithm whose generated rules are to be understood as default rules. This means that these rules must be tolerant to the negative information by allowing for the possibility of exceptions. The same classical algorithm is then used to learn recursively these exceptions.
We prove that the non-monotonic learning algorithm that realizes these ideas converges asymptotically to the concept to be learned. We also discuss various general issues concerning the problem of learning non-monotonic theories in the proposed framework.

1 Introduction

Over the last two decades research work in Artificial Intelligence has shown that many problems can not be captured fully within the realm of classical monotonic logic and that other logics which are non-monotonic are more suitable. In many cases these problems need a non-monotonic formalism if they are to be represented adequately. As a result if we want to develop methods for learning theories for these problems we need to develop learning formalisms that are based on non-monotonic representation (description) languages and concepts.

In this paper we will present a learning framework where the learned theories are non-monotonic. In particular, we will be interested in learning theories which are non-monotonic logic programs. We are thus interested in extending the language bias of the hypotheses from definite logic programs as in Inductive Logic Programming ([13]) to the language of non-monotonic logic programs.
The development of this non-monotonic learning framework will be done primarily within the context of learning from a set of examples hierarchical concepts with their exceptions. The particular learning problem that we will study within this framework is the usual problem of learning a concept from a set of positive and negative examples with emphasis on problems that can not be captured by classical definite logic programs. To do this we will exploit a recently proposed formalism for non-monotonic logic programming which although like all other such formalisms is based on the principle of negation as failure it does not rely on a NAF construct within the language but instead uses a limited form of classical negation. We are thus interested in extending the language bias from definite

logic programs to a more general class of logic programs that embodies both the non-monotonic principle of NAF as well as a form of explicit negation.

Within this formalism then the basic idea is to employ a classical learning algorithm to cover (as usual) the positive examples but with the difference that we are tolerant with respect to the negative information in the sense that we will allow a hypothesis generated by the classical algorithm even if this covers some of the negative examples. Hence a rule generated by the classical learning algorithm is now to be understood as a default rule in the sense that it holds in general but not always; it may have exceptions. In this way the logic program that we are building is not anymore a classical monotonic theory but now this constitutes a non-monotonic theory. If indeed it is the case that a generated rule also covers some of the negative examples the non-monotonic learning process can not stop but it will continue to address the problem of learning the concept that covers these negative examples i.e. the exceptions to the previously generated rule for the positive examples. This will be done by calling again the underlying classical algorithm that we are using in a dual form to learn the part of the negative concept that corresponds to these exception examples. As a result we will get a set of rules with the classical negation of the concept appearing in the head that imply the negation of the concept for these examples. These negative rules will then be added to the previously generated positive rules to form a new hypothesis.

Again we will allow anyone of these negative rules even if this covers some of the positive examples i.e. contains exceptions. The negative rules are also default rules and hence we need to be tolerant in the same way and allow for the possibility that they may cover some of the opposite (positive) examples. If this is so we will then iterate the above process by calling the classical algorithm to learn the "subpart" of the concept given by the subset of positive examples that form the exceptions to the negative rules generated in the previous step.

The problem of learning non-monotonic theories (logic programs) has already been address in the literature by several authors (e.g. [1], [11], [16]). In most of these works NAF is used in the object language of non-monotonic logic programs. This forces the need to invent new predicates (abnormality predicates) to handle exceptions. In [1, 11], exceptions are handled primarily at one level by some method of enumeration or with the use of inequality conditions. This makes it difficult to handle a hierarchy of exceptions (i.e. exceptions of exceptions etc).

Our approach is closely related to the algorithm presented in [16]. This is also capable of learning hierarchical concepts by using abnormality predicates and negation at the object level. In fact, this algorithm is, modulo the representation language, identical to the algorithm we present in this paper.[1] Our recursive treatment and the interpolation of the positive and negative information applies to that algorithm as well. Nevertheless, while in [16] the main task is to use the negative examples to learn the exceptions to the positive concept, we are also interested in learning (and using) the negative concept itself in a symmetric way to that of the positive concept. This task can be easily accommodated in the

[1] We thank the anonymous referees for pointing this connection out to us.

particular non-monotonic logic programming formalism that we are using.

The idea of learning the negative concept has already been studied in several earlier works. For example the work in CLINT ([5]) is concerned with learning the negative concept to be understood in a three valued logic. Our work has several points in common with this work but differs in the following ways. The main difference is the fact that here we are not only concerned with learning the negative concept per se but are also interested in a focused limited learning of parts of the negative concept as the need arises from the hierarchical nature of the positive concept. In this sense the two approaches are complementary in a way that our approach can adopt the other at the point where we need to learn the negative concept and conversely the extended CLINT system can adopt our method to learn hierarchical concepts. The two approaches also differ in the particular formalisms and description languages that they are using. Our approach with its tolerant rules seems to provide a simple framework in which non-monotonic learning algorithms can be developed.

Another method that is related to our work is that of [3]. This also uses a representation language which includes priorities but these are based on statistical information coming from the examples. Classical rules are generated for both the positive and negative concept and each one is attributed a statistical priority weight. However, like in [5], this method does not generate exceptions directly, as in our work and in [16], but exceptions come indirectly through the relative strength of the statistical weights of contradictory rules. Consequently, the theories generated in [3] will not have the multi-layered hierarchical structure that can be present in our theories. Again we believe that the two approaches can complement each other.

The rest of the paper is organised as follows. Section 2 gives some needed background information on the subject of non-monotonic logic programs. In section 3 we describe the non-monotonic learning algorithm and then show some of its main properties in section 4. Section 5 provides additional discussion on the non-monotonic learning algorithm and the issue of handling negative information.

2 Non-monotonic reasoning in Logic Programming

Non-monotonic reasoning in Logic Programming is done using the *negation as failure principle* (NAF) ([2]). This principle informally states that we can assume that some atom is false provided that it is not possible to derive the atom from our theory. Traditionally, in order to implement this principle a negation as failure operator is introduced in the language. Thus the language of Logic programming is extended from definite Horn clauses to general clauses of the form:

$$A \leftarrow L_1, \ldots L_n$$

where A is an atom and each L_i is either an atom A_i or a negation as failure literal *not* A_i. As usual each variable occurring in the clause is implicitly universally quantified. The semantics of this form of negation introduced in logic programs is given by some formalization of the NAF principle which can now be stated as:

not A holds iff A fails to hold

for any atomic goal A. Operationally, in order to satisfy a NAF goal, not p, an auxiliary computation is generated that checks that the goal, p, can not be derived. Consider for example the following program

$$\begin{gathered} fly(x) \leftarrow bird(x), not\ abnormal(x) \\ abnormal(x) \leftarrow penguin(x) \\ bird(x) \leftarrow penguin(x) \\ bird(Tweety) \end{gathered}$$

From the first clause (rule) of the program we can prove that Tweety flies if we can prove that Tweety is a bird and that *not abnormal(Tweety)* holds. The first of these follows directly from the last clause (rule) of the program whilst to show that *not abnormal(Tweety)* holds we need (following the NAF principle) to check that *abnormal(Tweety)* can not be proved from the program. Indeed, there is no way to prove this form the above program. On the other hand, if we add to the program the new clause *penguin(Tweety)* then we will be able to prove *abnormal(Tweety)* and thus *not abnormal(Tweety)* and *flies(Tweety)* do not hold.

Notice here that the effect of the NAF condition in the first rule is to prevent the conclusion of the rule to hold in the cases where we know (the program knows) that the abnormality property holds. In other words, the first rule can be understood as *x flies if x is a bird unless x is abnormal.* Thus the NAF operator "not" can informally be given the meaning of "unless" and we can thus see that NAF can help to capture exceptions to rules.

Various semantics have been proposed for general logic programs with NAF in their language ([2], [6], [15], [18]). In this paper we will adopt a recent new proposal ([8]) for capturing the NAF principle in Logic Programming that does not employ a NAF operator in the language but instead uses a limited form of classical negation together with a priority relation amongst the sentences of the program. The semantics of this framework, called the admissibility (or more generally the acceptability) semantics, is defined within an argumentation-based formalism and has been shown to be powerful enough to encompass most of the earlier semantics for NAF in Logic Programming.

In this framework the above logic program will be written equivalently as the following theory

$$\begin{gathered} fly(x) \leftarrow bird(x) \\ \neg fly(x) \leftarrow penguin(x) \\ bird(x) \leftarrow penguin(x) \\ bird(Tweety) \end{gathered}$$

with a (partial) ordering relation between the sentences that assigns the second rule higher than the first. (In the subsequent chapters this will be denoted by

a directed arc between the rules.) Here "$\neg$" denotes explicit negation and can be used to represent negative information about the world that the program is modelling. From this theory we can conclude that Tweety flies because we can derive this from the first rule and there is no way to derive $\neg fly(Tweety)$ from the program. If we add the sentence $penguin(Tweety)$ then we can derive both $fly(Tweety)$ and $\neg fly(Tweety)$ from the program understood as a classical theory. But in the non-monotonic admissibility semantics of the theory the second conclusion overrides the first since the part of the program that derives $\neg fly(Tweety)$ contains the second rule which is designated higher than the first rule which belongs to the part of the program that derives the first conclusion $fly(Tweety)$.
Note that the language bias given by this framework contains together with the NAF principle that it embodies a form of explicit (classical) negation.

In general, this argumentation-based framework for non-monotonic Logic Programs is defined as follows.

Definition 1. *(Background logic)*
Formulae in the language $\mathcal{L}$ of the framework are defined as $L \leftarrow L_1, \ldots, L_n$, where $L, L_1, \ldots, L_n$ are positive or explicit negative literals. The only inference rule is the classical modus ponens rule

$$\frac{L \leftarrow L_1, \ldots, L_n \qquad\qquad L_1, \ldots, L_n}{L} \qquad (n \geq 0) \qquad \Box$$

We assume that, together with the set of sentences $\mathcal{T}$, we are given a priority relation $<$ on these sentences (where $\phi < \psi$ means that ϕ has lower priority than ψ). The role of the priority relation is to encode locally the relative strength of rules in the theory, typically between contradictory rules. We will require that $<$ is irreflexive and antisymmetric.

Definition 2. *(Non-Monotonic Theory or Logic Program)*
A theory $(\mathcal{T}, <)$ is a set of sentences $\mathcal{T}$ in $\mathcal{L}$ together with a priority relation $<$ on the sentences of $\mathcal{T}$.

We now proceed to define a notion of attack on these theories based on the possible conflicts that we can have in a theory $\mathcal{T}$ between a literal L and its explicit negation $\neg L$ and on the priority relation $<$ on $\mathcal{T}$.

Definition 3. *(Attacks)*
Let $(\mathcal{T}, <)$ be a theory and $T, T' \subseteq \mathcal{T}$. Then T' attacks T iff there exists L, $T_1 \subseteq T'$ and $T_2 \subseteq T$ such that

(i) $T_1 \vdash_{min} L$ and $T_2 \vdash_{min} \neg L$
(ii) $(\exists r' \in T_1, r \in T_2 \text{ s.t. } r' < r) \Rightarrow (\exists r' \in T_1, r \in T_2 \text{ s.t. } r < r')$. $\Box$

$T \vdash_{min} L$ means that $T \vdash L$ under the background logic and that L can not be derived from any proper subset of T. Using this we then define the basic notion of an admissible subset of a given theory (program).

Definition 4. *(Admissibility)*
Let $(\mathcal{T}, <)$ be a theory and $T \subseteq \mathcal{T}$. Then T is *admissible* iff T is consistent and for any $T' \subseteq \mathcal{T}$ if T' attacks T then T attacks T'.

The (sceptical) semantics of a theory can then be defined in terms of its admissible subsets as follows.

Definition 5. Let $(\mathcal{T}, <)$ be a theory and L a ground literal. Then L is a *non-monotonic consequence* of the theory iff L holds in every maximal admissible subset of $\mathcal{T}$.

It can be shown that given a logic program P, with NAF in its object language we can define a corresponding equivalent theory $\mathcal{D}(P)$. This transformation is motivated from the interpretation of *not p* as *unless p*. For example, if we have a rule "$p \leftarrow q, not\ r$" then this transformed into two sentences "$p \leftarrow q$" and "$\neg p \leftarrow r$", and the second is assigned higher priority than the first.

Finally, we mention that for most of the work in this paper we will be concerned with a subclass of non-monotonic logic programs in this formalism. These will be theories where their set of contradictory rules (i.e. rules with opposite conclusions) can be separated into classes where the rules in each class are totally ordered by the priority relation of the theory. In such theories in order to decide if an atom, A, holds we need to show that A can be derived classically using some rule, r, for A and that $\neg A$ can not be derived (classically) using some rule r' which is designated higher than the rule r by the priority relation on the program.

3 Description of the Algorithm

The main learning problem we are interested in is the usual problem of learning a concept from a set of positive and negative examples. Hence given a background theory and a set of examples we want to generate a hypothesis within the language bias of non-monotonic logic programs described above that covers all the positive examples and does not cover any of the negative examples. In particular, we are interested in problems which can not be learned within the language bias of definite (classical) logic programs. For example these concepts may have an inherent hierarchical nature that cannot be captured by classical logic programs. In these cases in addition to the coverage of the examples by the generated hypothesis we also want to identify the hierarchical structure of concept. As a result of this the generated hypothesis (non-monotonic logic program) will also have as consequence the negation of the concept for some of the negative examples. In other words, part of the negative concept will also be learned.

The basic idea behind the algorithm that we are proposing is to use a classical learning algorithm in a tolerant way in the sense that the hypothesis computed to cover a set of positive examples may also cover some of the negative examples. We say that, given a background theory B, a hypothesis H covers an example

e iff $B \cup H$ implies l, such that $|l| = e$, where $|l| = e'$ or $|l| = \neg e'$ where e' is an atom. If the example e is negative (resp. positive) and $B \cup H$ implies e (resp. $\neg e$) then this example is called an exception to H.

The set of negative examples that are covered by this generalization are considered to be an exception to this generalization, and the algorithm then attempts to learn this exception by using the classical algorithm on this set of examples. This procedure is applied iteratively on subsets of the original examples until a point is reached where, under the admissibility semantics, all the positive examples are covered and no negative example is covered. We illustrate the main features of the algorithm via the next example.

Example 1. Consider the background theory B
$bird(x) \leftarrow penguin(x)$
$superpenguin(x) \leftarrow penguin(x)$
$bird(a), bird(b), penguin(c), penguin(d), superpenguin(e), superpenguin(f)$
Consider also the set of examples $E = E^+ \cup E^-$ where $E^+ = \{flies(a), flies(b), flies(e), flies(f)\}$ and $E^- = \{flies(c), flies(d)\}$.
□

The algorithm in the first step will attempt to cover all the positive examples by calling a classical learning algorithm. This can be done by the rule $flies(x) \leftarrow bird(x)$. But this rule also covers the negative examples. These will then be considered to be exceptions to it, and a new phase will begin that will now attempt to learn the concept that these exceptions may form. The set of positive examples for the classical algorithm in this phase will be $E_1^+ = E^- = \{flies(c), flies(d)\}$. Since the concept of the exception should not cover any of the positive examples covered by the rule, the negative examples for this phase will be the set $E_1^- = E^+$. Suppose that the classical algorithm will now compute the rule $\neg flies(x) \leftarrow penguin(x)$. This new rule will be related to the one of the previous step via a link that gives priority to $\neg flies(x) \leftarrow penguin(x)$ over $flies(x) \leftarrow bird(x)$.

Note that we are learning the negative concept $\neg flies$ in exactly the same way as the positive concept $flies$. Again some of the examples in E_1^- are covered by the new rule. These are the "exceptions of the exceptions", and the algorithm again calls the classical learning algorithm to generate a rule with input examples $E_2^+ = \{flies(e), flies(f)\}$ and $E_2^- = E_1^+$. This rule is $flies(x) \leftarrow superpenguin(x)$ which is added to the theory together with a link from itself to the rule $\neg flies(x) \leftarrow penguin(x)$. This last rule does not cover any of examples in E_2^- and the procedure terminates. The result is the hypothesis H

$$\begin{tikzcd}
\text{flies(x)} \leftarrow \text{bird(x)} \\
\neg\,\text{flies(x)} \leftarrow \text{penguin(x)} \arrow[u, bend left] \\
\text{flies(x)} \leftarrow \text{superpenguin(x)} \arrow[u, bend left]
\end{tikzcd}$$

The hypothesis H together with the background knowledge B cover, under the admissibility semantics, all the examples in E^+ and do not cover any of the examples in E^-.

The ideas presented so far can be captured formally in the following algorithm. Let P be the set of positive examples, N the set of negative examples, B the background knowledge, H the current hypothesis and R a rule initialized to be empty.

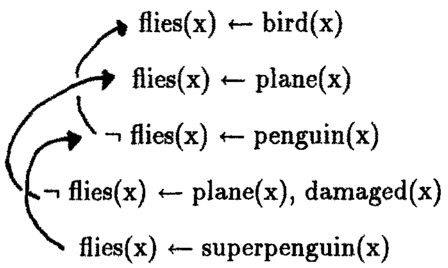
```
Algorithm NMLearn($P, N, B, R, H$)
Begin
Classic-Learn($P, N, B, H'$)
if $E \subset E'$ , where $E = \{e|e \in N$ and $e$ covered by $H'\}$ and
$E' = \{e|e \in N$ and $e$ covered by $R\}$
then

    begin
    $H = H \cup H'$; add priority links from each rule in $H'$ to $R$;
    $M_i = \{e_i|e_i \in N$ and covered by rule $R'_i \in H'\}$.
    For each $M_i <> \emptyset$ do
    $NMLearn(M_i, P, B, R'_i, H'')$
    end

else enumerate the exceptions of $R$ and terminate;
end
```

Some comments on the algorithm are in order. First note that is possible that at each step more than one rule may be needed to cover a set of the examples. Consider the following example.

Example 2. Let the background knowledge be $B' = B \cup \{plane(g), plane(h), plane(k), plane(m), damaged(k), damaged(m)\}$, where B is the background knowledge of example 1. Moreover let the set of examples be the same with those of example 1 augmented with the positives $flies(g), flies(h)$ and the negatives $flies(k), flies(m)$. Then the algorithm will compute the hypothesis



□

Notice that the algorithm computes the hypothesis in a depth first manner. Sets of rules computed at each level are explicitly associated with a rule of the previous level and are considered to express the part of the negative concept that forms of the exception to the rule. Moreover, it is obvious from the recursive nature of the algorithm *NMLearn* that the treatment of positive and negative examples is symmetric. At each odd level (starting from level one) a set of positive examples is generalized while at each even level set of negative ones.

Another important issue is the treatment of the negative examples by the *Classic-Learn* algorithm. No special restriction is put on the rules that this algorithm returns. Nevertheless, it is clear that the procedure may also cover some of the negative examples. The non-monotonic algorithm can tolerate such a coverage, since in the subsequent steps these exceptions will be captured and introduced in the final hypothesis. We should though stress the fact that the amount of tolerance that the algorithm exhibits is an important issue to be discussed further in section 5.

4 Properties of the Algorithm

A learning problem is formally stated as a tuple $(\vdash, LB, LE, LH)$, where $\vdash$ is a correct provability relation, LB the language of the background knowledge, LE the language of examples and LH the hypothesis language. Then we are given a background knowledge $B \in LB$ and a set of positive and negative examples $E = E^+ \cup E^-$, $E \in LE$. The problem is to find a hypothesis $H \in LH$ such that $B, H \vdash E^+$ and $\forall e \in E^-$, $B, H \not\vdash e$. In this paper we are concerned with two particular learning problems. The first is defined by the tuple $(\vdash_C, LB, LE, LH)$ and the second by the tuple $(\vdash_N, LB, LE, NLH)$. The provability relation $\vdash_C$ refers to a correct provability relation for first order Horn clauses, while the relation $\vdash_N$ to a correct provability relation for non-monotonic logic programs under the admissibility semantics. Furthermore, LB and LH is the language of definite logic programs, while NLH denotes the language of non-monotonic logic programs. Finally, LE is a language of ground atoms.

In order to prove the formal results of this section we will assume that any input set of examples can be covered by the *Classic-Learn* algorithm with a single clause and so *NMLearn* uses only a single clause at each level. Hence, the NHL language is now restricted to non-monotonic theories of the form $H = (H_1, \ldots, H_n)$, where each H_i is a single rule and the links are of the form $H_{i+1} > H_i$ for $1 \leq i < n$. Furthermore, since in this section we are interested in learning hierarchies the language bias also imposes the restriction that $B \models \forall(\text{ body}(H_{i+1}) \rightarrow \text{body}(H_i))$ for $1 \leq i < n$, where B is the given background knowledge. We call this language bias the ***hierarchy language bias.*** Combining this with the single clause bias we get the ***single hierarchy language bias.***

Furthermore, assume that the algorithm *Classic-Learn* always computes a clause $c(x) \leftarrow b_{min}(x)$ that covers all the examples for which the following minimality condition holds: For any other clause $c(x) \leftarrow b(x)$ that also covers the same set of examples, $B \models \forall x(b_{min}(x) \rightarrow b(x))$ holds.[2] We call such a clause a ***minimal clause*** or rule. A minimal clause together with the background

[2] Here we identify the variables of the two rules $c(x) \leftarrow b_{min}(x)$ and $c(x) \leftarrow b(x)$ by

knowledge entails a set of conclusions that is a subset of the consequences of any other clause that covers the positive examples at hand.

This minimality condition implies that the hypotheses computed by *NMLearn* comply with the single hierarchy langauge we have assumed in this section. This is a straightforward consequence of the observation that for any H_{i+1} in the hypothesis $H = (H_1, \ldots H_m)$, H_{i+1} is a minimal rule that covers a set of examples E_{i+1}, which are the exceptions to the rule H_i and so H_i is a rule that also covers the examples E_{i+1}.

Next we prove that the procedure *NMLearn* always terminates.

Proposition 6. *The algorithm NMLearn terminates on any finite set of examples that is given as input.*

Proof. Let E be a finite set of examples given as input to the algorithm. We will show that at each iteration of the algorithm the classical algorithm is asked to cover less examples than at the previous step, and therefore, since E is finite, the algorithm terminates.

Let E_i^+ be the set of positive examples to be covered at step i (i odd) by a clause r_i. Suppose also that r_i covers a non-empty set of negative examples E_i^-. Otherwise, if $E_i^- = \emptyset$ then the algorithm terminates. This set E_i^- will be covered by a new (negative) rule r_{i+1}. At this step the algorithm either terminates by enumeration[3] when $E_{i+1}^+ \not\subset E_i^+$ or if $E_{i+1}^+ \subset E_i^+$ it continues to find a rule r_{i+2} to cover E_{i+1}^+. □

Proposition 7. *The depth of the hypothesis computed by the algorithm NMLearn on a set of examples $E = E^+ \cup E^-$ is minimum in the sense that there is no other hypothesis that identifies the concept and has smaller depth.*

Proof. Let M be an optimal hypothesis wrt the depth such that $M \vdash_N E^+$ and $M \not\vdash_N e$ for every $e \in E^-$. The algorithm in the first step has to cover the set E^+ with the minimal clause of E^+, say r_1. Due to the restricted single hierarchy language bias of *NLH* the same set of examples has also to be covered by of one of the rules of M, say r'_1. If r_1 does not cover any examples from E^- is optimal. If not, then since r_1 is minimal, and so $B \models$body$(r_1) \rightarrow$body(r'_1), every negative example that is covered by r_1 is also covered by r'_1. This means that if the depth is increased by the *NMLearn* by one, then the depth in M is increased by one as well. Also if we denote by $E_1'^-$ the set of negative examples that have to be covered by the optimal hypothesis in the next level note that $E_1'^- \supseteq E_1^-$, where E_1^- is the set of negative examples that have to be covered by the algorithm *NMLearn*. Since *NMLearn* computes the minimal clause of E_1^- we see that

identifying the corresponding variables in the heads of the rules. We assume that the heads do not contain any non-variable terms. If this is so, we first need to homogenise such rules by introducing explicit equality conditions in the body of the rule.

[3] Note that the additional condition of minimality on *Classic* − *Learn* ensures that at each iteration $E_{i+1}^+ \subseteq E_i^+$, hence enumeration is only required if $E_{i+1}^+ = E_i^+$.

if during this step the algorithm covers some positive examples then the same positive examples will be covered by the optimal hypothesis and hence if the depth needs to be increased further by the algorithm this will also be true in M. Iterating these argument see that M has the same depth as the hypothesis computed by the algorithm. □

We will now show that the algorithm $NMLearn$ identifies in the limit ([7], [9]) the learning problem $(\vdash_N, LB, LE, NLH)$, under some restrictions on the input examples.

Definition 8. Let LE be the language of examples, $E_\infty = \{e_1, e_2, \ldots\}$ denote an infinite stream of examples, such that $e_i \in LE \times \{+, -\}$ and E_∞ a complete enumeration of LE. Let also $E^+_\infty = \{e| < e, + > \in E_\infty\}$ denote the positive examples and $E^-_\infty = \{e| < e, - > \in E_\infty\}$ the negative ones.
We say that the background knowledge B is **partially suitable** for E_∞ and LH if $\exists H \in LH$ such that $B, H \vdash E^+_\infty$. We also say that the background knowledge B is **suitable** for E_∞ and LH iff it is partially suitable and $\forall e \in E^-_\infty, B, H \not\vdash e$.
We say that a learning problem $(\vdash, LB, LE, LH)$ is **partially identified in the limit** by an algorithm $LEARN$ iff for every stream E_∞ and partially suitable background knowledge $B \in LB$, $LEARN$ accepts incrementally E_∞ and there exists a step i for which $\forall j \geq i : H_i = H_j$ and $B, H_i \vdash E^+_\infty$. □

Definition 9. A learning problem $(\vdash, LB, LE, LH)$ is **identified in the limit** by an algorithm $LEARN$ iff for every stream $E_\infty = E^+_\infty \cup E^-_\infty$ and suitable background knowledge $B \in LB$, $LEARN$ accepts incrementally E_∞ and there exists a step i for which $\forall j \geq i : H_i = H_j$ and $B, H_i \vdash E^+_\infty$, $\forall e \in E^-_\infty$, $B, H_i \not\vdash e$ holds. □

The following definition is used in the next theorem.

Definition 10. A set of examples $E = E^+ \cup E^-$ is called **strictly consistent** wrt a background knowledge B, if there are no subsets, $E^+_i \subseteq E^+$ and $E^-_j \subseteq E^-$ such that the minimal rules H_i of E^+_i and H_j of E^-_j have equivalent bodies under B, i.e. $B \models \forall\ (\text{body}(H_i) \leftrightarrow \text{body}(H_j))$. □

Theorem 11. *If the learning problem $(\vdash_C, LB, LE, LH)$ is partially identified in the limit by the algorithm Classic-Learn, then the learning problem $(\vdash_N, LB, LE, NLH)$ on any strictly consistent infinite set of examples E_∞ is identified in the limit by the algorithm $NMLearn$.*

Proof. Let $E_\infty = E^-_\infty \cup E^+_\infty$. Then since *Classic-Learn* partially identifies in the limit E_∞ then after some step it will stabilize and there will be a rule H_i such that $\forall j \geq i : H_i = H_j$ and $B, H_i \vdash E^+_\infty$. We call H_i H^1_S. If H^1_S does not cover any negative example then the concept is identified. If not then at some point the negative examples will be encountered and the second level of the algorithm will be activated. So again because *Classic-Learn* partially identifies in the limit the classical learning problem, at some point after the stabilization of the first

rule a second negative rule H_S^2 will be stabilized that will imply all the negative examples that H_S^1 covers. Note that this second rule will identify in the limit the infinite stream of examples $E_2 = E_2^+ \cup E_2^-$, where $E_2^+ = \{e | B \cup H_S^1 \vdash_C e, e \in E_1^-\}$ and $E_2^- = E_\infty - E_2^+$, where $E_1^- = E_\infty^-$. Iterating these arguments we can construct a sequence of rules $H_S^1, H_S^2, \ldots H_S^m$ each of which partially identifies a learning problem on an infinite stream of examples $E_i = E_i^+ \cup E_i^-$ where $E_i^+ = \{e | B \cup H_S^{i-1} \vdash_C e, e \in E_{i-1}^-\}$ and $E_i^- = E_\infty - E_i^+$. Such a finite sequence will not be computed by the algorithm only if the algorithm reaches a point where an enumeration is required or if an unbounded sequence of rules is needed in order to identify the set E_∞. The first case can not occur as E_∞ is strictly consistent. If the second case occurs then note that due to the optimality of the depth of the hypothesis constructed by the *NMLearn* algorithm (proposition 7) no other finite set of rules can cover E_∞. So the background knowledge and the set E_∞ are not suitable and hence there is nothing to prove for algorithm.
Assume now that *NMLearn* computes a finite set of rules $H = \{H_1, \ldots, H_m\}$ and terminates without enumeration. Let $E_j^{+/-}$ be the set of examples covered minimally by the rule H_j. We first prove that for every $e \in E_\infty^+$, $B, H \vdash_N e$ (the background knowledge B will be omitted from the premises for the rest of the proof). Let E_{2k+1}^+ be the last set on an odd level in which e occurs. If $2k+1 = m$ we see that $H_m \vdash_C e$ and hence $H_N \vdash e$. Assume that $2k+1 \neq m$. If $H_{2k+2} \not\vdash_C e$ then for all $2n > 2k+2$, $H_{2n} \not\vdash_C e$ and then $H \vdash_N e$. If $H_{2k+2} \vdash_C e$ then by construction of *NMLearn*, E_{2k+1}^+ can not be the last set in which e occurs. Hence $H \vdash_N e$, for every $e \in E_\infty^+$.
We next prove that if $e \in E_\infty^-$, then $H \not\vdash_N e$. Assume the contrary and let H_{2k+1} be the last rule on an odd level, such that $H_{2k+1} \vdash_C e$. If $2k+1 = m$ then the *NMLearn* will compute a new rule H_{m+1} such that $H_{m+1} \vdash_C e$. This gives a contradiction since H_m is the last such rule. Let $2k+1 < m$. Then by construction $H_{2k+2} \vdash_C e$. Furthermore, $H_{2k+3} \not\vdash_C e$ since H_{2k+1} is the last rule on an odd level, such that $H_{2k+1} \vdash_C e$. Hence $H \not\vdash_N e$. □

5 Further Discussion

The convergence result presented in the previous section is restricted to the case of strictly consistent sets of examples. Sets of examples that are not strictly consistent reveal important issues for practical systems that address the problem of learning exceptions, both on an algorithmic as well as a conceptual level. The following example is illustrative.

Example 3. Consider the background knowledge $B = \{bird(a), feathered(a), bird(a'), feathered(a'), bird(b), light(b), bird(b'), light(b'), bird(c), feathered(c), broken-wings(c), bird(c'), feathered(c'), broken-wings(c'), bird(d), light(d), big(d), bird(d'), light(d'), big(d')\}$, and the examples $E = E^+ \cup E^-$ where $E^+ = \{flies(a), flies(b), flies(a'), flies(b')\}$ and $E^- = \{flies(c), flies(d), flies(c'), flies(d')\}$. □

In this example the minimal clause of both E^+ and E^- is $flies(x) \leftarrow bird(x)$. The algorithm *NMLearn* of the previous section will terminate by enumerating the exceptions of the rule $flies(x) \leftarrow bird(x)$. Note however that the following hypothesis:

$$\begin{tikzcd}
 & \text{flies(x)} \leftarrow \text{bird(x), feathered(x)} \\
\neg\, \text{flies(x)} \leftarrow \text{bird(x), feathered(x), broken-wings(x)} \arrow[ur, bend left] & \\
 & \text{flies(x)} \leftarrow \text{bird(x), light(x)} \\
 & \neg\, \text{flies(x)} \leftarrow \text{bird(x), light(x), big(x)} \arrow[u, bend left]
\end{tikzcd}$$

covers all the positive examples and none of the negative without the need for any enumeration. In this case (see also example 3.2) it is clear that the single hierarchy language bias is too restrictive as these examples are inherently multi-hierarchical and thus the algorithm is forced to enumerate. In general to avoid enumeration in the *NMLearn* algorithm whenever it reaches a stage where $E = E'$ holds for R and H', the algorithm can backtrack to the point where R was computed to generate a set of new rules to replace R which are more specific than R in the sense that they have less consequences than those of R. One possible way to do this is to employ the standard "covering approach" where we split the set of examples that we want to cover into smaller subsets and try to cover each of these separately.

The negative input information present in the negative examples can be used in two different ways during the learning process of the algorithm *NMLearn*. On the one hand they can be used within the classical algorithm employed by *NMLearn* to specialise the monotonic generalization rules that this generates. On the other hand, they can also be used as examples that define a part of the negative concept which forms an exception. We thus see that negative information can play a double role, either that of specialization or that of exception. Clearly each one of these roles on its own is not optimal. In the case where the only role is that of specialization we may have situations (e.g. hierarchical concepts) where it is not possible to find an appropriate theory. On the other hand, if we treat all negative information as exception then we run the risk of been overgeneral and thus overtolerant as the above example 3 shows. In general, a learning system must find a balance between the two different uses of the negative information. One possible strategy to achieve this balance in the algorithm *NMLearn* is to allow the *Classic-Learn* to use the negative examples to specialise as much as possible (without losing coverage of any of the positive examples) and then treat the remaining negative examples as exceptions.

Finally, note that it may happen that some of the negative examples are used neither in the specialization process by the *Classic-Learn* algorithm nor as exceptions by *NMLearn*. Those examples remain essentially unused. Hence it is reasonable to require that the learning process should try to learn the part of the negative concept implied by these examples. In fact, this learning of the negated concept is necessary in some situations. Consider the following example.

Example 4. Let $B = \{Republican(a), Republican(b), Quacker(c), Quacker(d)$ and $E = E^{+} \cup E^{-}$, where $E^{+} = \{Pacifist(c), Pacifist(d)\}$ and $E^{-} = \{Pacifist(a), Pacifist(b)\}$. □

The obvious generalization is the clause $Pacifist(x) \leftarrow Quacker(x)$. However, note that from this theory we can entail $Pacifist(m)$ for any m such that $Republican(m)$ and $Quacker(m)$ holds although intuitively in view of the negative examples we do not expect this. If we require that we also learn the concept that generalizes the unused negative examples, the clause $\neg Pacifist(x) \leftarrow Republican(x)$ will be also computed. Then within this new theory under the admissibility semantics, the value of $Pacifist(m)$ is undefined.

This learning of the negative concept can easily be accommodated within our framework by requiring that after *NMLearn* terminates on the original input of examples a second run is activated to learn input of examples a second run is activated to learn the negative examples unused in the first run. This will give a second (dual) hierarchy whose rules are not related to these of the first hierarchy and thus allow for the possibility of many admissible extensions. As a result of this, under the skeptical semantics, some literals may become undefined. Note that in this way we combine together the non-monotonic learning of a hierarchical concept, as in [16], with learning the negative concept as in [3] and [5].

6 Conclusions-Further work

We have proposed a framework for learning non-monotonic logic programs and have studied within this framework the problem of learning a hierarchical concept from a set of examples. The main features of our approach are (i) the exploitation of a classical learning algorithm that is tolerant to the negative information, (ii) the handling of exceptions through learning the relevant part of the negative concept and (iii) the ability to complete within the same framework the learning of the negative concept. These features have been naturally accommodated within the particular non-monotonic logic programming formalism that we have adopted that embodies together with the NAF principle a form of explicit negation.

Our approach is closely related to that of [16]. Nevertheless, we argue that a further step may be useful if the task is to learn the negative concept completely. This can be easily accommodated within the particular non-monotonic formalism used in this paper. The definition of the negative concept through explicit negation provides a more expressive representation language where undefinetness can be accommodated. The idea of learning the negative concept and thus allowing a three valued semantics, was also used in [5] and [3]. However, in these works the negative concept is not related so strongly to the positive one in the sense that they do not generate explicit hierarchical structures for the concept to be learned. Thus our work provides a framework that links together these works [3], [5], [16] where one can investigate how these different approaches can complement each other. We are planning to test empirically the theoretical work developed here, compare these results with those of [3], [5] and investigate how we can link our approach to these other methods in order to get better results. We also plan to investigate to what extend our techniques can be applied to problems of refining classifications learned by artificial neural network systems.

Other possible directions for further research are the following. One issue is the investigation of the dual nature of negative information for specialization and exceptions as pointed out in section 5.
Another issue is the extension of the non-monotonic learning algorithm along two main directions. The first is to modify the algorithm with incremental learning capabilities ([4]). Preliminary investigations indicate that this can be done in a way analogous to *NMLearn*. Also incremental specialization techniques (e.g. [11], [17]) may be useful in developing incremental variants of the algorithm. The second extension that needs further study is the problem of learning with a background knowledge which is itself non-monotonic. We are currently investigating how classical algorithms such as those based on inverse resolution ([12]) can be modified to work on non-monotonic background knowledge.

Acknowledgements We thank the anonymous referees for many valuable comments. This work was partly supported by the ESPRIT BRA project Compulog 2 no 6810.

References

1. M. Bain and S. Muggleton, Non-monotonic learning. In: J.E. Hayes-Michie and E. Tyugu, eds., *Machine Intelligence 12*. Oxford University Press, 1990
2. K.L. Clark. Negation as failure. In *Logic and databases*, Gallaire and Minker, eds., Plenum Press, 1978.
3. J. Cussens, A. Hunter and A. Srinivasan. Generating explicit ordering for non-monotonic logics. *Proc. of AAAI-93*.
4. L. De Raedt. *Interactive Theory Revision: an Inductive Logic Programming Approach*. Academic Press, 1992.
5. L. De Raedt and M. Bruynooghe. On negation and three-valued logic in interactive concept learning. *Proc. of the 9th European Conference on AI, ECAI-90*, 207-212, 1990.
6. M. Gelfond and V. Lifschitz. The stable model semantics for logic programs. *Proc. of the 5th International Conference and Symposium on Logic Programming*, 1070-1080, MIT Press, 1990.
7. E.M. Gold. Language identification in the limit. *Information and Control*, 10:447-474, 1967.
8. A. Kakas, P. Mancarela and P. M. Dung. The acceptability semantics for logic programs. *Proc. of 11th Inter. Conference on Logic Programming, ICLP-94*, 504-519, MIT Press, 1994.
9. J-U. Kietz and S. Dzeroski. Inductive logic programming and learnability. *SIGART Newsletters*, **5(1)**, 1994.
10. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

11. C. Ling. Non-Monotonic specialization. *Proc. of the Inductive Logic Programming Workshop, ILP-91*, 1991.
12. S. Muggleton and W. Buntime. Machine invention of first order predicates by inverting resolution. *Proc. of the 5th Inter. Conference on Machine Learning*, 339-352, Kaufmann, 1988.
13. S. Muggleton. Inductive logic programming. *New Generation Computing*, *8*, 295-318, 1991.
14. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. submitted.
15. T. Przymusinski, On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning*, 5, 167-205, 1989.
16. A. Srinivasan, S. Muggleton and M. Bain. Distinguishing exceptions from noise in non-monotonic learning. *Proc. of the International Workshop on Inductive Logic Programming*, S. Muggleton and K. Furukawa, Japan, 1992.
17. S. Wrobel. On the proper definition of minimality in specialization and theory revision. *Proc. of the European Conference on Machine Learning, ECML-93*, Vienna, 1993, LNAI 667, Springer Verlag.
18. A. Van Gelder, K. A. Ross and J. S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *Proc. of the 7th Symposium on Principles of Database Systems, PODS-88*, 221-230, ACM Press, 1988.