

# Nonmonotonic Abductive Inductive Learning

Oliver Ray

*Department of Computer Science, University of Bristol,  
Merchant Venturers Building, Woodland Road, Bristol BS8 1UB, UK*

---

## Abstract

Inductive Logic Programming (ILP) is concerned with the task of generalising sets of positive and negative examples with respect to background knowledge expressed as logic programs. Negation as Failure (NAF) is a key feature of logic programming which provides a means for nonmonotonic commonsense reasoning under incomplete information. But, so far, most ILP research has been aimed at Horn programs which exclude NAF, and has failed to exploit the full potential of normal programs that allow NAF. By contrast, Abductive Logic Programming (ALP), a related task concerned with explaining observations with respect to a prior theory, has been well studied and applied in the context of normal logic programs. This paper shows how ALP can be used to provide a semantics and proof procedure for nonmonotonic ILP that utilises practical methods of language and search bias to reduce the search space. This is done by lifting an existing method called Hybrid Abductive Inductive Learning (HAIL) from Horn clauses to normal logic programs. To demonstrate its potential benefits, the resulting system, called XHAIL, is applied to a process modelling case study involving a nonmonotonic temporal Event Calculus (EC).

---

## 1 Introduction

Inductive Logic Programming (ILP) [38] is a branch of Artificial Intelligence (AI) concerned with the generalization of positive and negative examples with respect to prior background knowledge expressed in a logic program formalism. Compared to other AI representations, logic programs are expressive and easy for humans to understand. Moreover, Negation as Failure (NAF) [6] gives logic programming a nonmonotonic inference mechanism for reasoning with defaults and exceptions under incomplete information. Since incompleteness is an inherent feature of any learning problem, effective utilization of NAF is potentially a major strength of the ILP paradigm.

---

*Email address:* oray@cs.bris.ac.uk (Oliver Ray).

To date, most ILP research has been aimed at Horn programs that exclude NAF. While several approaches have been proposed for normal programs with NAF, as in [4,19,7,25,28,5,51,13,39,1,43,49,56] and more recently in [50,40,12], these impose strong restrictions on the learning setting or they lack efficient strategies for guiding the computation. As a result, practitioners are forced to apply Horn systems to normal problems, for which they are not intended, often with no guarantees of soundness or completeness and usually without the ability to fully exploit NAF. For example, by various transformations [33,31,41,42] Horn systems can be made to learn action theories in which NAF is used to model the persistence of properties through time. But the limitations of such approaches, discussed later, only highlight the need to develop semantically and procedurally well-founded nonmonotonic learners.

This work aims to realise a practical ILP approach that generalises successful techniques of language and search bias from Horn clauses to normal programs. It is based on the premise that such biases are even more essential in the nonmonotonic case, where the search space is much larger and traditional pruning techniques are not applicable. The proposal is to exploit techniques from a closely related branch of AI, called Abductive Logic Programming (ALP) [21], which grew from efforts to provide a semantics and proof procedure for NAF [8]. Although simple forms of abductive reasoning have been used in ILP for learning rules with predicates not defined by the examples [37,32], the correspondence between ALP and NAF can potentially be exploited to even greater effect by enabling the learning of normal logic programs.

This paper shows how a method called Hybrid Abductive Inductive Learning (HAIL) [44], which integrates ALP and ILP in a common reasoning framework, can be lifted from Horn clauses to normal programs. The technique is based on the construction and generalization of a preliminary ground hypothesis called a Kernel Set [44] that bounds the search space in accordance with user specified language and search bias. The result is a three-stage process where abduction and deduction are used, respectively, to compute the head and body literals of a Kernel Set, which is then generalised by a subsumption-based inductive search technique. This paper shows how these three stages can all be specified as executable ALP tasks in order to provide a formal semantics and concrete proof procedure, called XHAIL, for nonmonotonic ILP.

The rest of the paper is structured as follows. Section 2 introduces the key notation, terminology, and background material. Section 3 shows how each phase of the XHAIL procedure can be formalised as a nonmonotonic ALP task. Section 4 discusses some of the theoretical and practical implications. Section 5 illustrates XHAIL with a biologically motivated process modelling case study. Section 6 compares the approach with related work and Section 7 concludes. This paper extends the abstract in [45] with a more detailed description of the method, a larger case study, and further discussion of related work.

## 2 Background

This section introduces the key notation and terminology. Section 2.1 reviews some logic programming definitions [27], recalls the stable model semantics [15] for normal programs, and outlines the task of ALP [21]. Section 2.2 describes the task of ILP [38] and introduces two popular forms of language and search bias called mode declarations and compression [36]. Section 2.3 summarises the original HAIL approach [44] for learning Horn programs.

### 2.1 Abductive Logic Programming (ALP)

This paper assumes a standard first order language whose terms are defined in the usual way. An atom is a predicate  $p$  (of arity  $n$ ) followed by an  $n$ -tuple of terms  $(t_1, \dots, t_n)$ . A literal is either an atom  $a$  (positive literal) or the negation of an atom  $not\ a$  (negative literal). A (normal) clause is an expression of the form  $a \leftarrow l_1, \dots, l_m$  where  $a$  is an atom (called the head atom) and the  $l_i$  are literals (called body literals). A fact is a clause of the form  $a \leftarrow \top$  (often abbreviated to just  $a$ ) where  $\top$  is an atom denoting logical truth. A constraint is a clause of the form  $\perp \leftarrow l_1, \dots, l_m$  (often abbreviated to just  $\leftarrow l_1, \dots, l_m$ ) where  $\perp$  is an atom denoting logical falsity. A (logic) program is a set of clauses. A clause or program is Horn iff all of its literals are positive and is normal otherwise. A (Herbrand) interpretation is a set of ground atoms. An interpretation  $I$  satisfies a positive (resp. negative) ground literal  $l = a$  (resp.  $l = not\ a$ ) iff  $a \in I$  (resp.  $a \notin I$ ). It satisfies a set of ground literals iff it satisfies each ground atom in the set; and it satisfies a ground clause iff it satisfies the head atom or fails to satisfy at least one body literal.

A (Herbrand) model  $M$  of a program  $P$  is an interpretation  $I$  that satisfies every ground instance of every clause  $C$  in  $P$ . A model  $M$  is minimal if no strict subset is also a model. Moreover, it is stable if  $M$  is the unique minimal model of the Horn program  $P^M$  obtained from the ground instances of  $P$  by removing all clauses with a negative literal not satisfied in  $M$  and removing all negative literals from the remaining clauses. A program  $P$  entails a set of ground literals  $L$  (under the credulous stable model semantics), denoted  $P \models L$ , if at least one stable model of  $P$  satisfies  $L$ . A goal, or query,  $G$  is a set of literals  $L = \{l_1, \dots, l_n\}$  that will usually be written  $?l_1, \dots, l_n$ . A clause  $C$  is said to ( $\theta$ -)subsume a clause  $D$  iff there is a substitution  $\theta$  such that the head atom of  $D$  is the head atom of  $C\theta$  and each body literal of  $D$  is in  $C\theta$ . A program  $P$  is said to ( $\theta$ -)subsume a program  $Q$  iff for each clause  $C_i \in P$  there is a clause  $D_i \in Q$  such that  $C_i$  subsumes  $D_i$  and  $D_i \neq D_j$  for all  $i \neq j$  (i.e.,  $P$  is obtained from  $Q$  by replacing terms in  $Q$  with variables and/or by dropping individual literals or whole clauses from  $Q$ ).

ALP seeks to find the conditions under which a query  $G$  (goal) can be made to succeed from a program  $T$  (theory) composed of facts, rules and constraints. The ALP task is usually defined as computing a set of ground atoms  $\Delta$ , called an explanation, together with a ground substitution  $\theta$ , called an answer substitution, such that  $T \cup \Delta \models G\theta$ . The atoms in  $\Delta$  are usually restricted to a set  $A$  of predicates called abducibles, which identify concepts for which only partial information is available in the theory (e.g., potential faults in a diagnosis task or possible actions in a planning domain). Any pair  $(\Delta, \theta)$  which satisfies these properties is called an abductive solution of  $G$  wrt.  $T$  and  $A$ .

Hereafter,  $\phi(T, G, A)$  will denote the set of all abductive solutions of  $G$  wrt.  $T$  and  $A$ . Systems for computing such abductive solutions can be classed into two complementary approaches: top-down ALP methods that extend resolution inference with the ability to assume literals [21], and bottom-up methods based on model construction techniques like Answer Set Programming (ASP) [26]. To ensure termination of these procedures, it is assumed some appropriate bound is placed on computational resources (e.g., maximum resolution depth in an ALP system or total execution time in an ASP system).

## 2.2 Inductive Logic Programming (ILP)

ILP seeks to find a program  $H$  (hypothesis) that generalises a set of literals  $E$  (examples) wrt. a program  $B$  (background knowledge). In this paper,  $B$  and  $H$  are normal programs, while  $E$  is a set of ground literals (with positive and negative literals representing positive and negative examples, respectively). Given  $B$  and  $E$  as inputs, the task of ILP is to compute a set of clauses  $H$  such that  $B \cup H \models E$ .<sup>1</sup> The clauses in  $H$  are usually constrained by some form of language and search language bias. This paper employs two popular forms of bias called mode declarations and compression, both taken from [36].

A mode declaration  $m$  is either a head declaration  $modeh(r, s)$  or a body declaration  $modeb(r, s)$ , where  $s$  is a ground literal, the scheme, which serves as a template for literals in the head or body of a hypothesis clause, and  $r$  is an integer, the recall, which limits how often the scheme is used. An asterisk  $*$  denotes an arbitrary recall. A scheme can contain special placemaker terms of the form  $\#type$ ,  $+type$  and  $-type$ , which stand, respectively, for ground terms, input terms and output terms of a predicate  $type$ . The distinction between input and output terms is that any input term in a body literal must be an input term in the head or an output term in some preceding body literal.

<sup>1</sup> In the Horn case, this can be written  $B \cup H \models e^+$  for all  $e^+ \in E^+$  and  $B \cup H \not\models e^-$  for all  $e^- \in E^-$  where  $E^+$  and  $E^-$  are the sets of unnegated and negated atoms in  $E$ , respectively. In the Horn case, it can also be written  $B \cup H \models P$  and  $B \cup H \cup N \not\models \perp$  where  $P = \{e \leftarrow \top \mid e \in E^+\}$  and  $N = \{\perp \leftarrow e \mid e \in E^-\}$ .

Each set  $M$  of mode declarations is associated with a set of clauses  $\mathcal{L}(M)$ , called the language of  $M$ , such that  $C = a \leftarrow l_1, \dots, l_n \in \mathcal{L}(M)$  iff the head atom  $a$  (resp. each body literal  $l_i$ ) is obtained from some head (resp. body) declaration in  $M$  by replacing all  $\#$  placemarkers with ground terms and all  $+$  (resp.  $-$ ) placemarkers with input (resp. output) variables. If  $m$  is any mode declaration, then  $pred(m)$  denotes the predicate  $p$  at the front of the scheme  $s$ , while  $schema(m)$  denotes the literal obtained from  $s$  by replacing all placemarkers with distinct variables  $X_1, \dots, X_n$ , and  $type(m)$  denotes the sequence of literals  $t_1(X_1), \dots, t_n(X_n)$  such that  $t_i$  is the type of the placemaker replaced by the variable  $X_i$ . If  $M$  is a set of mode declarations, then  $M^+$  and  $M^-$  denote the head and body declarations in  $M$ , respectively.

As well as language bias, ILP hypotheses are also constrained by search bias. A successful method, described in [36], involves maximising a compression score obtained by subtracting the number of literals in the hypothesis from the number of examples covered. This helps to avoid over-fitting by preferring the simplest hypothesis that explains the examples. Another common approach, also explained in [36], involves using the recall of a mode declaration during hypothesis construction to bound the number of times its scheme can be used (with the same input terms) and using the type of a placemaker to determine which terms can replace it (in any ground instance).

### 2.3 Hybrid Abductive Inductive Learning (HAIL)

HAIL is a mode-directed ILP approach that integrates abductive, deductive and inductive reasoning in a common learning framework. Given a background theory  $B$ , examples  $E$ , and mode declarations  $M$ , it aims to return a highly compressive hypothesis  $H$ , in the language of  $M$ , that entails  $E$  wrt.  $B$ . In the Horn case [44], hypotheses are constructed incrementally by a covering loop designed to generalise one selected example, called a seed example, at a time. Partial hypotheses are successively formed until all examples are covered. For each seed example  $e \in E$ , this is done by constructing and generalising a preliminary ground hypothesis,  $K$ , called a Kernel Set of  $B$  and  $e$ .

Intuitively, a Kernel Set is a maximally specific explanation of the selected seed example. It is constructed and generalised by a three-phase methodology. The first phase returns a set of head atoms  $\Delta$  (abductive explanations) that entails  $e$  wrt.  $B$ . The second phase then adds to each head atom a set of body atoms (deductive consequences) entailed by  $B$  — to give a Kernel Set  $K$ . The third phase finds a compressive hypothesis  $H$  (inductive generalisation) that subsumes  $K$ . The main challenge is to exploit language and search bias when constructing and generalising  $K$ . To do this, HAIL uses a multi-clause extension of a method called Mode Directed Inverse Entailment (MDIE) [36].

As formalised below, the key idea underlying the HAIL approach is to use the abduced literals to seed the formation of an inductive hypothesis.

- **abductive phase:** first HAIL computes a set of ground facts

$$\Delta = \left\{ \begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_n \end{array} \right\}$$

such that  $B \cup \Delta \models e$  and each atom  $\alpha_i$  in  $\Delta$  is a well-typed ground instance of a clause in the language  $\mathcal{L}(M^+)$  of the head declarations  $M^+$  (as defined in Section 2.2). These atoms are computed by an abductive procedure that returns an explanation  $\Delta$  of the seed example  $e$  wrt. the program  $B$  using the type information specified by the head declarations.

- **deductive phase:** then HAIL computes a set of ground clauses

$$K = \left\{ \begin{array}{c} \alpha_1 \leftarrow \delta_1^1, \dots, \delta_1^{m_1} \\ \vdots \\ \alpha_n \leftarrow \delta_n^1, \dots, \delta_n^{m_n} \end{array} \right\}$$

such that  $B \models \delta_i^j$  for all  $1 \leq i \leq n, 1 \leq j \leq m_i$  and each clause  $\alpha_i \leftarrow \delta_i^1, \dots, \delta_i^{m_i}$  in  $K$  is a well-typed ground instance of a clause in  $\mathcal{L}(M)$ . These atoms are computed by a deductive procedure that finds the successful ground instances of the queries obtained by substituting a set of input terms into the + placemarkers of the body declaration schemas.

- **inductive phase:** finally, HAIL computes a set of clauses

$$H = \left\{ \begin{array}{c} a_1 \leftarrow d_1^1, \dots, d_1^{m'_1} \\ \vdots \\ a_{n'} \leftarrow d_{n'}^1, \dots, d_{n'}^{m'_{n'}} \end{array} \right\}$$

such that  $B \cup H \models E$ . This is done by generalising each Kernel Set clause, in turn, using a top down A\*-like search of lattice of clauses which subsume  $K$  to find a maximally compressive clause that is in the language  $\mathcal{L}(M)$  of  $M$  and is consistent with the other clauses in  $H$ .

While MDIE is the basis of many popular Horn systems, such as Progol [36,37] and Aleph [54], HAIL is shown to be a sound extension of this method that increases the class of problems soluble in practice [44].

### 3 eXtended Hybrid Abductive Inductive Learning (XHAIL)

This section lifts the HAIL methodology from Horn theories to normal logic programs. As before, the aim is to make effective use of language and search bias to guide the computation by constructing and generalising a preliminary ground Kernel Set. The main challenge is that of generalising the abductive, deductive, and inductive phases to handle NAF. The approach proposed below is based on representing each phase as an abductive task in order to exploit the correspondence between abduction and negation [8]. This has the benefits of providing a standard abductive semantics [22] for each phase of the procedure and allows all three phases to be implemented using an off-the-shelf and high-performance abductive reasoning engine or answer set solver.

The extended procedure, called XHAIL, is specified in Figure 1 and explained below. The inputs are a background theory  $B$ , examples  $E$ , mode declarations  $M$ , and an integer depth bound  $d$  (which, as in Progol and Aleph, bounds the number of body literals through which any term is linked to the head atom). The output is a compressive hypothesis  $H$  that falls within the language of  $M$  and entails  $E$  wrt.  $B$ . Each hypothesis is computed in three phases:  $P_1$ ,  $P_2$  and  $P_3$ . As detailed below, each of these phases  $P_i$  can be defined (in the notation of Section 2.1) as an abductive task  $\phi(T_i, G_i, A_i)$  with its own theory  $T_i$ , goal  $G_i$ , and abducibles  $A_i$ . Each phase will now be described and illustrated with the following running example from [50]:

$$B = \left\{ \begin{array}{l} bird(X) \leftarrow penguin(X) \\ bird(a) \\ bird(b) \\ bird(c) \\ penguin(d) \end{array} \right\}$$

$$E = \left\{ \begin{array}{l} flies(a) \\ flies(b) \\ flies(c) \\ not\ flies(d) \end{array} \right\}$$

$$M = \left\{ \begin{array}{l} modeh(*, flies(+bird)) \\ modeb(*, penguin(+bird)) \\ modeb(*, not\ penguin(+bird)) \end{array} \right\}$$

While the focus of this paper is on the formal specification of the three phases of the extended HAIL procedure, some discussion is also provided, in Section 4, regarding their implementation using ALP and ASP technologies. The whole procedure is further exemplified on a more substantial case study in Section 5.

### 3.1 Abductive Phase

The first phase,  $P_1$ , computes a set of ground atoms  $\Delta = \bigcup_{i=1}^n \alpha_i$  such that  $B \cup \Delta \models E$  and each  $\alpha_i$  is a well-typed instance of a clause in  $\mathcal{L}(M^+)$ . This is a straightforward abductive task. Since each abduced atom will go in the head of a Kernel Set clause, the abducibles  $A_1$  are obtained from the set of predicates  $p/n$  appearing at the front of some head declaration scheme. But, to ensure any abduced atoms satisfy the required bias, for each such predicate, a clause is created of the form  $p(X_1, \dots, X_n) \leftarrow p^*(X_1, \dots, X_n), p'(X_1, \dots, X_n)$  containing two fresh predicates  $p^*/n$  and  $p'/n$ . In effect,  $p'$  is just an abducible proxy for  $p$ , while  $p^*$  identifies the ground instances of  $p$  that satisfy the head declarations. For each head declaration  $m \in M^+$ , a clause is created whose head atom is obtained by starring the predicate in  $schema(m)$  and whose body atoms are those in  $type(m)$ . These additional clauses allow to distinguish abduced instances of  $p$  from implied instances of  $p$ ; and they ensure all (minimal) solutions are well-typed instances of the mode declarations. As shown in Figure 1, a set of head atoms is obtained from each explanation  $W$  by replacing any occurrence of  $p'$  with  $p$ . Each set of atoms  $\Delta$  produced in this way is then generalised by the deductive and inductive phases, below.

**Example:** In the running example,  $E$  contains three positive examples and one negative example which must all be explained.  $M^+$  contains a single head declaration  $m = modeh(*, flies(+bird))$ , so that  $pred(m) = flies/1$  and  $schema(m) = flies(X)$  and  $type(m) = bird(X)$ . Thus, there is one abducible predicate  $flies'/1$  and one type predicate  $flies^*/1$  defined by the two clauses  $flies(X) \leftarrow flies^*(X), flies'(X)$  and  $flies^*(X) \leftarrow bird(X)$ . The former states that it is possible to assume  $X$  flies if  $X$  has the correct type; and the latter states that  $X$  has the correct type if  $X$  is a bird. Note that, if there had been other head declarations for  $flies/1$ , these would have contributed alternative definitions to  $flies^*/1$ . But, as there are not, only one abductive explanation exists in this case: i.e.,  $W = \{flies'(a), flies'(b), flies'(c)\}$ . Thus, replacing each new predicate  $p'$  by the original predicate  $p$ , gives

$$\Delta = \left\{ \begin{array}{l} flies(a) \\ flies(b) \\ flies(c) \end{array} \right\}$$

**INPUTS:** logic program  $B$  (background), ground literals  $E$  (examples),  
mode declarations  $M$ , and positive integer  $d$  (depth bound)

**% — Abductive Phase ( $P_1$ ) — %**

**let**  $A_1$  be the set of predicates containing one fresh predicate  $p'/n$  for each  
predicate  $p/n = \text{pred}(m)$  in the scheme of some head declaration  $m \in M^+$

**let**  $T_1$  be the set of clauses obtained by adding to  $B$  one clause  
 $p(X_1, \dots, X_n) \leftarrow p'(X_1, \dots, X_n), p^*(X_1, \dots, X_n)$  for each predicate  $p'/n \in$   
 $A_1$  and one clause  $p^*(t_1, \dots, t_n) \leftarrow r_1(v_1), \dots, r_q(v_q)$  for each head declara-  
tion  $m \in M^+$  with schema  $p(t_1, \dots, t_n)$  and types  $r_1(v_1), \dots, r_q(v_q)$

**let**  $W$  be **any** explanation in  $\phi(T_1, E, A_1)$

**let**  $\Delta$  be the set of ground facts obtained by replacing each abduced atom  
 $p'(t_1, \dots, t_n) \in W$  with the corresponding fact  $p(t_1, \dots, t_n)$

**% — Deductive Phase ( $P_2$ ) — %**

**let**  $A_2$  be the empty set of predicates  $\emptyset$

**let**  $T_2$  be the program obtained by adding to  $B$  each fact in  $\Delta$

**for** each fact  $\alpha_i \in \Delta$

**let**  $m_i$  be **any** head declaration in  $M$  whose schema subsumes  $\alpha_i$

**set**  $n_i$  to the set of terms in  $\alpha_i$  corresponding to  $+$  placemarkers in  $m_i$

**set**  $k_i$  to the fact  $\alpha_i$

**repeat** up to  $d$  times

**let**  $Q$  be the set of goals  $?type(m)\sigma, schema(m)\sigma$  where  $m \in M^-$  is a  
body declaration and  $\sigma$  is a substitution binding all input variables  
(i.e., all variables that replaced a  $+$  placemarkers) in  $m$  to a term in  $n_i$

**let**  $R$  be the set of ground literals of the form  $schema(m)\sigma\theta$  where  
 $schema(m)\sigma$  appears in a goal  $G \in Q$  and  $\theta$  is an answer substitu-  
tion in  $\phi(T_2, G, A_2)$

**add** to the body of  $k_i$  all literals in  $R$  (not already in  $k_i$ )

**add** to  $n_i$  all (new) terms in  $R$  corresponding to  $-$  placemarkers

**let**  $k'_i$  be the clause obtained from  $k_i$  by replacing all distinct terms  
corresponding to  $+$  and  $-$  placemarkers with fresh variables

**let**  $K'$  (resp.  $K$ ) be the set of clauses  $\{k'_1, \dots, k'_n\}$  (resp.  $\{k_1, \dots, k_n\}$ )

**% — Inductive Phase ( $P_3$ ) — %**

**let**  $A_3$  be the singleton set of predicates  $\{use/2\}$

**let**  $T_3$  be the program obtained by adding to  $B$  one clause  $\alpha'_i \leftarrow$   
 $use(i, 0), try(i, 1, \delta''^1_i) \dots try(i, m_i, \delta''^{m_i}_i)$  for each  $k'_i = \alpha'_i \leftarrow \delta^1_i \dots \delta^{m_i}_i \in$   
 $K'$  and two clauses  $try(i, j, \delta''^j_i) \leftarrow use(i, j), \delta^j_i$  and  $try(i, j, \delta''^j_i) \leftarrow$   
 $not\ use(i, j)$  for each literal  $\delta^j_i$  in the clause  $k'_i$  with variables  $\delta''^j_i$

**let**  $U$  be **any** explanation in  $\phi(T_3, E, A_3)$

**let**  $H$  be the program obtained from  $K'$  by removing every body atom  $\delta^j_i$  for  
which the abducible  $use(i, j)$  is not in  $U$ , and removing every clause whose  
head atom  $\alpha'_i$  does not have a corresponding atom  $use(0, i)$  in  $U$

**OUTPUT:** logic program  $H$  (hypothesis)

Fig. 1. XHAIL Specification

### 3.2 Deductive Phase

The second phase,  $P_2$ , computes a ground program  $K = \bigcup_{i=1}^n \alpha_i \leftarrow \delta_i^1, \dots, \delta_i^{m_i}$  where  $B \cup \Delta \models \delta_i^j$  for all  $1 \leq i \leq n, 1 \leq j \leq m_i$  and each clause  $\alpha_i \leftarrow \delta_i^1, \dots, \delta_i^{m_i}$  is a well-typed ground instance of a clause in  $\mathcal{L}(M)$ . To do this, each head atom is saturated with body literals using a nonmonotonic generalisation of the Progol level saturation method [36]. For this, the abductive system is made to behave as a deductive query answering procedure by declaring an empty set  $\emptyset$  of abducibles. Each head atom is then processed by choosing a head declaration  $m_i$  to initialise a growing set of input terms  $n_i$  which are substituted into the  $+$  placemarkers of the given body declarations  $M^-$  to generate a set  $Q$  of goals  $G$  whose successful ground instances, obtained from  $\phi(B \cup \Delta, G, \emptyset)$ , result in a set of literals  $R$  which can be added into the body of the clause  $k_i$  with head atom  $\alpha_i$ . Additionally, any new output terms are inserted into  $n_i$ . The clause  $k_i$  resulting from each abducible  $\alpha_i$  is then added to the Kernel Set  $K$ .

**Example:** In the running example,  $\Delta$  contains three atoms that must each be generalised (in turn or in parallel). The first one,  $\alpha_1 = \textit{flies}(a)$ , is subsumed by the schema of just one head declaration  $m_1 = \textit{modeh}(*, \textit{flies}(+bird))$  giving one initial input term  $a$ . The atom  $\textit{flies}(a)$  is initialised to the head of the clause  $k_1$  and the input term  $a$  is substituted into the schema of the first available body declaration  $m = \textit{modeb}(*, \textit{penguin}(+bird))$ , with schema  $\textit{penguin}(X)$  and type  $\textit{bird}(X)$ , to give the goal  $?bird(a), \textit{penguin}(a)$  using a substitution  $\sigma$  that binds the input variable  $X$  to the input term  $a$ . Conversely, the other body declaration gives the goal  $?bird(a), \textit{not penguin}(a)$ . Since only the latter goal succeeds (with the empty answer substitution  $\theta = \emptyset$ ), the literal  $\textit{not penguin}(a)$  is added to the body of  $k_1$ . As no new output terms are introduced by  $\theta$ , no more goals are formed and no other body literals are added to  $k_1$ . Thus, processing all of the head atoms in this way, gives

$$K = \left\{ \begin{array}{l} \textit{flies}(a) \leftarrow \textit{not penguin}(a) \\ \textit{flies}(b) \leftarrow \textit{not penguin}(b) \\ \textit{flies}(c) \leftarrow \textit{not penguin}(c) \end{array} \right\}$$

Whereupon, replacing all input and output terms by fresh variables, gives

$$K' = \left\{ \begin{array}{l} \textit{flies}(X) \leftarrow \textit{not penguin}(X) \\ \textit{flies}(Y) \leftarrow \textit{not penguin}(Y) \\ \textit{flies}(Z) \leftarrow \textit{not penguin}(Z) \end{array} \right\}$$

### 3.3 Inductive Phase

The third phase,  $P_3$ , computes a compressive theory  $H = \bigcup_{i=1}^{n'} a_i \leftarrow d_i^1, \dots, d_i^{m'_i}$  that subsumes  $K$  and entails  $E$  wrt.  $B$ . This is done by deleting from  $K'$  as many literals (and clauses) as possible while ensuring correct coverage of the examples. The abductive system is prepared for this by a transformation involving two fresh predicates  $try/3$  and  $use/2$ . For every clause  $k'_i \in K'$ , each body literal  $\delta'^j_i$  is reduced to its variables  $\delta''^j_i$  and wrapped inside an atom of the form  $try(i, j, \delta''^j_i)$ . Then, an extra atom  $use(i, 0)$  is added to the body of  $k'_i$  and two clauses  $try(i, j, \delta''^j_i) \leftarrow use(i, j), \delta'^j_i$  and  $try(i, j, \delta''^j_i) \leftarrow not\ use(i, j)$  are created for each  $\delta'^j_i \in k'_i$ . These clauses are added to the theory along with those in  $B$ . Then, putting  $use/2$  as the only abducible means each explanation  $U$  of the examples  $E$  will be a set of ground atoms  $use(i, j)$  indicating that the corresponding literals  $\delta'^j_i$  from  $K'$  should be kept in  $H$  while all of the other literals from  $K'$  should be deleted.

The literal  $use(i, j)$  literally means ‘use’ the  $j$ th literal in the  $i$ th clause of  $K'$  (where 0 is the head atom and  $1..m_i$  are the body atoms). The intuition is that, in order for a head atom  $\alpha'_i$  from  $K'$  to contribute towards the satisfaction of an example  $e$  in  $E$ , each of the body atoms  $try(i, j, \delta'^j_i)$  must be satisfied. Thanks to the two rules added for this atom, its truth can be ensured in one of two ways: by simply assuming  $not\ use(i, j)$ ; or by abducing  $use(i, j)$  and proving  $\delta'^j_i$ . The former effectively ignores  $\delta'^j_i$  as if it were not there, while the latter solves  $\delta'^j_i$  as if it had been part of the clause. Similarly, the atom  $use(i, 0)$  determines if the  $i$ th clause is included in the hypothesis, or not.

**Example:** In the running example,  $K'$  contains three clauses that must be generalised. The first gives rise to the three transformed clauses shown below

$$\begin{aligned} flies(X) &\leftarrow use(1, 0), try(1, 1, X) \\ try(1, 1, X) &\leftarrow not\ use(1, 1). \\ try(1, 1, X) &\leftarrow use(1, 1), not\ penguin(X). \end{aligned}$$

The other clauses in  $K'$  produce nearly identical transformations, which can in fact be omitted (since variants of the same clause in  $K'$  can be merged at the only risk of increasing the size of the smallest  $\Delta$  from which a given  $H$  may be computed). Either way, all of the minimal abductive explanations, such as  $\{use(1, 0), use(1, 1)\}$ , result in the same final hypothesis

$$H = \left\{ flies(X) \leftarrow not\ penguin(X) \right\}$$

## 4 Discussion

The extended HAIL procedure differs from its monotonic predecessor in some key respects which are designed to avoid the unsoundness and incompleteness that would result if standard Horn clause pruning heuristics and incremental covering methods were applied in the normal setting where hypotheses intended to cover earlier examples could be invalidated by hypotheses intended to cover later examples. For this reason, XHAIL has the ability to process all of the examples in  $E$  in one go and to generalise all of the clauses in  $K$  at the same time. This also means that the system is not dependent on the order of the examples and will always be able to find a globally optimal hypothesis with maximum compression.

Another key aspect of XHAIL is that the body literals of a Kernel Set  $K$  are not entailed by the theory  $B$  alone (as in earlier versions of HAIL), but also by the explanation  $\Delta$  of the examples  $E$ . This is significant because it overcomes an inherent incompleteness of MDIE shown by a famous problem [57] of finding the hypothesis  $H = \{odd(s(X)) \leftarrow even(X)\}$  for examples  $E = \{odd(s(s(s(0))))\}$ ,  $not\ even(s(s(s(0))))\}$  and theory  $B = \{even(0)\} \cup \{even(s(X)) \leftarrow odd(X)\}$ . Given declarations  $M = \{modeh(*, odd(s(+)))\} \cup \{modeb(*, even(+))\}$  this task cannot be solved by mainstream systems like Progol or Aleph because more than one instance of the hypothesis is needed to prove the example [57]. But this hypothesis is easily returned by XHAIL as it subsumes the Kernel Set  $K = \{odd(s(0)) \leftarrow even(0)\} \cup \{odd(s(s(0))) \leftarrow even(s(s(0)))\}$ . While one existing Horn clause system [14] and two full clausal systems [59,18] have been developed to avoid this incompleteness in monotonic logic programs, none of them support NAF and they are all considerably more complex than XHAIL, even on this simple example.

As in the task above, a Kernel Set  $K$  often has the property that  $B \cup K \models E$ . But this is not a requirement of  $K$ , as its real purpose is to provide a syntactic and semantic bias that delimits a region of the search space likely to contain hypotheses. Strictly speaking, this role is not played by  $K$  but by the non-ground theory  $K'$  which (even in the Horn case) may not be a hypothesis (as it could violate integrity). In any case, the search procedure is responsible for finding a correct generalisation by dropping literals as necessary. For example, if  $B = \{b \leftarrow e\}$  and  $E = \{e\}$ , then  $K = \{e \leftarrow b\}$  is now a Kernel Set of  $B$  and  $e$ , but the only hypothesis which subsumes  $K$  and which would be returned by XHAIL is  $H = \{e\}$ . Of course, it is possible to ensure that  $B \cup K \models E$  by requiring all the body literals in  $K$  to be satisfied in a model of  $B$  consistent with  $\Delta$  (as described in [45] previously). But this would have the effect of including in  $K$  all facts now believed false, excluding from  $K$  all facts now believed true, and would also sacrifice completeness of the new procedure.

The semantic correctness of XHAIL follows directly from the soundness of the abductive procedure used to compute  $\phi(T_3, E, A_3)$  in the inductive phase,  $P_3$ , which ensures  $T_3 \cup U \models E$ , where  $U$  is a set of ground atoms of the form  $use(i, j)$ . This means there is a stable model  $I$  of  $T_3 \cup U$  which satisfies  $E$ . It can then be shown that the subset  $J \subseteq I$  obtained by removing from  $I$  all atoms with the predicates  $use/2$  and  $try/3$  is a stable model of  $B \cup H$  that satisfies  $E$ . Thus,  $B \cup H \models E$ . Syntactic correctness follows partly from the inductive phase,  $P_3$ , which ensures each hypothesis clause  $h_i \in H$  is a subset of some clause  $k'_i \in K'$ , and partly from the abductive phase,  $P_2$ , which ensures  $k'_i$  is in the language  $\mathcal{L}(M)$  of  $M$ . The linking of input and output variables is not strictly enforced in  $H$  but, if necessary, can be ensured through the introduction of additional constraints stating that certain literals may not be used unless some other literals are.

Many abductive systems have a built-in preference for explanations with fewer abducibles. If such a bias is used by XHAIL's abductive engine, it will favour Kernel Sets with fewer clauses and will return hypotheses with fewer literals. However, because some maximally compressive hypotheses, especially those involving some form of recursion, may not subsume a minimal cardinality Kernel Set, an iterative deepening strategy is desirable — at least in the abductive phase. It should also be remarked that the approach is only complete for computing minimal hypotheses with finite Herbrand models. For instance, the task of finding the hypothesis  $H = \{int(s(X)) \leftarrow int(X)\}$  for the example  $E = \{not\ max\}$  and theory  $B = \{max \leftarrow int(X), not\ int(s(X))\} \cup \{int(0)\}$  would technically require an infinite Kernel Set.

Two implementations of the XHAIL procedure have been evaluated: the first one using a top-down ALP system to perform the abduction; and the second one using an bottom-up ASP solver as the computational engine. Preliminary experiments suggest the latter ASP approach achieves greater performance on problems that utilise numerical functions, integrity constraints, and cyclic or recursive definitions; but that the former ALP approach can be applied more easily to problems that exploit list operations, standard Prolog libraries, and potentially infinite domains.

While a full description of the implementation is beyond the scope of this paper, it is worth remarking that the results in the next section were obtained using an ASP version of XHAIL with an iterative deepening strategy that computes Kernel Sets having progressively more clauses until a subsuming hypothesis is found. ASP was used because it was found to outperform ALP in this domain. Iterative deepening was used because experience shows that the number of abductive explanations (and hence the number and size of possible Kernel Sets) grows quickly with the number of abducibles [46,2]; but there are usually very few minimal explanations and these are the ones which tend to produce compressive theories that do not overfit the examples.

## 5 Case Study

This section illustrates the XHAIL procedure on a nonmonotonic learning task that uses a temporal formalism called the Event Calculus (EC) [24] to induce a simple model of metabolic regulation for the bacterium *E. coli* [20]. This study reveals many advantages of XHAIL: including its ability to learn hypotheses for predicates not in the examples (i.e., it performs non-Observation Predicate Learning [37]) and its ability to reason through logical cycles with negated atoms (i.e., it handles locally unstratified programs [3]) while making extensive use of language and search bias to bound the search space.

### 5.1 Inputs

The EC formalism used in this example is a logical framework for representing and reasoning about states, actions, and time [35]. For convenience, this study adopts a well-known logic programming formulation, known as the Simplified Event Calculus (SEC) [53], which includes three sorts of terms: *time-points*, denoted in this paper by integers; *events*, denoting actions that happen at various times; and *fluents*, denoting properties that hold at various times. There are three axioms (A1-A3) which govern the way events affect fluents. As formalised in Figure 2, these axioms exploit NAF to model the persistence of fluents when not affected by any known initiating or terminating events.

The first axiom states a fluent  $F$  holds at time  $T$  if an event  $E$  happened at an earlier time  $S$  that initiated  $F$  (i.e., caused  $F$  to become true) and no intervening event clipped  $F$  (i.e., terminated  $F$ , thereby causing it to become false). The second axiom states that a fluent  $F$  is clipped between times  $S$  and  $T$  if an event  $E$  happens at some intermediate time  $R$  that terminates  $F$ . The third axiom states that a fluent  $F$  holds at time  $T$  if  $F$  was initially true (i.e., was true at time 0) and no intervening event clipped  $F$ . The fluents and events, along with their initiating and terminating conditions, are defined in order to represent the following highly simplified model of *E. coli* metabolism.

In brief, *E. coli* is a well studied micro-organism that lives in the human gut. Ordinarily, this bacterium prefers to feed on the simple sugar glucose but, if necessary, it can feed on the complex sugar lactose by producing extra enzymes that break down lactose into glucose. But, to conserve energy, *E. coli* has evolved an efficient control mechanism which ensures these extra enzymes are only produced when lactose is available as a food source but glucose is not. The object of the exercise is to infer the existence of this mechanism from observations describing how the availability of lactose and glucose varies in response to the addition of these sugars to the growth medium.

```

% — Background Theory (B) — %
holdsAt(F,T) ← happens(E,S), S<T, initiates(E,F,S), not clipped(S,F,T). (A1)
clipped(S,F,T) ← happens(E,R), S<R, R<T, terminates(E,F,R). (A2)
holdsAt(F,T) ← initially(F), not clipped(0,F,T). (A3)
time(0..9). (T1)
sugar(lactose ; glucose). (T2)
event(add(G) ; use(G)) ← sugar(G). (T3)
fluent(available(G)) ← sugar(G). (T4)
initiates(add(G), available(G), T) ← sugar(G), time(T). (D1)
terminates(use(G), available(G), T) ← sugar(G), time(T). (D2)
← happens(use(G),T), not holdsAt(available(G), T). (D3)
happens(add(lactose), 0). (N1)
happens(add(glucose), 0). (N2)

% — Examples (E) — %
holdsAt(available(lactose), 1). (E1)
holdsAt(available(lactose), 2). (E2)
not holdsAt(available(lactose), 3). (E3)

% — Mode Declarations (M) — %
modeh(*, happens(use(#sugar),+time). (M1)
modeb(*, holdsAt(#fluent,+time)). (M2)
modeb(*, not holdsAt(#fluent,+time)). (M3)

```

Fig. 2. XHAIL Inputs (B, E & M)

```

% — Hypothesis (H) — %
happens(use(glucose),T) ← holdsAt(available(glucose),T). (H1)
happens(use(lactose),T) ← holdsAt(available(lactose),T),
not holdsAt(available(glucose),T). (H2)

```

Fig. 3. XHAIL Output (H)

The ontology of the domain is formalised by type axioms (T1-T4). The first axiom states the time-line consists of the integers 0 through 9 (where the notation ‘..’ is a shorthand for an integer range). The second axiom states that lactose and glucose are both sugars (where the notation ‘;’ is a shorthand for alternative arguments). The third axiom states that for each sugar,  $G$ , there are two events,  $\text{add}(G)$  and  $\text{use}(G)$ . The former denotes the action performed by a scientist when he adds  $G$  to the growth medium; while the latter denotes the action performed by the bacteria when it uses  $G$  as its food source. The fourth axiom states that for each sugar,  $G$ , there is one fluent,  $\text{available}(G)$ , which refers to the availability of  $G$  as a food source.

Now, suppose a scientist conducts an experiment on a culture of *E. coli*, to which he adds lactose and glucose and measures the availability of lactose in order to infer the conditions under which this bacterium uses these sugars. Some initial knowledge is represented by the domain axioms (D1-D3). The two rules state that adding a sugar  $G$  to the medium (resp. using up  $G$ ) initiates (resp. terminates) the availability of  $G$ ; and the constraint states it is impossible for *E. coli* to use a sugar which is not available. The two narrative events (N1-N2) formalise the actions of adding the two sugars at the beginning of the experiment. The three example observations (E1-E3) formalise the observation of lactose availability over the next three time-points.

As yet, no information has been given about the situations in which *E. coli* might use the sugars lactose and glucose. Instead, our purpose is to learn this from the examples  $E$  in (E1-E3) and the background theory  $B$  consisting of the temporal axioms (A1-A3), type axioms (T1-T4), domain axioms (D1-D3), and narrative events in (N1-N2). To this end, the mode declarations (M1-M3) state that hypothesized clauses may have atoms of the form  $\text{happens}(\text{use}(g), T)$  in their head, where  $g$  is a constant (denoting a sugar) and  $T$  is a variable (denoting a time); and may also have literals of the form  $\text{holdsAt}(f, T)$  and  $\text{not holdsAt}(f, T)$  in their bodies where  $f$  is a constant (denoting a fluent) and  $T$  is a variable (denoting a time that is mentioned in the head atom).

Given the inputs  $B$ ,  $E$  and  $M$  in Figure 2, the XHAIL proof procedure in Figure 1 will construct a compressive hypothesis  $H$  in the language of  $M$  that entails  $E$  wrt.  $B$ . In this case, XHAIL returns just one hypothesis  $H$  which, as formalised in Figure 3, states that *E. coli* will use glucose whenever glucose is available, and that it will use lactose whenever lactose is available but glucose is not. There is only one solution because the observed change in the availability of lactose at time 3 must be due the fact that lactose is used at time 2 but not at time 1. However, the only way time-points 1 and 2 can be distinguished is by an implicit change in the availability of glucose, which must be used at time 1 but not at time 2. (Note that an initiating or terminating effect on a fluent is always felt at the time-point immediately after an event.)

## 5.2 Output

The XHAIL system used in this case study employs an iterative deepening strategy to automatically construct and generalise Kernel Sets of progressively increasing size until a solution is found. This is achieved by exploiting the built-in preference of the abductive engine to compute minimal explanations. As shown below, several alternative Kernel Sets may therefore have to be considered before a hypothesis is finally returned.

The head atoms of each Kernel Set are computed by the abductive phase  $P_1$ . Here, there is one abducible  $happens'/2$  in  $A_1$  associated with the predicate  $happens/2$  in the only head declaration. Thus,  $B$  is supplemented with two typing clauses in  $T_1$ : i.e.,  $happens(E, T) \leftarrow happens'(E, T)$ ,  $happens'(E, T)$  and  $happens'(use(G), T) \leftarrow sugar(G), time(T)$ . There is one singleton set  $W$  containing the fact  $happens'(use(lactose), 2)$  which results in a minimal  $\Delta$  composed of a single head atom  $happens(use(lactose), 2)$ .

The body literals of a Kernel Set are computed by the deductive phase  $P_2$ . Here,  $A_2$  is empty and one atom  $happens(use(lactose), 2)$  is added to  $B$  in  $T_2$ . There is one term  $2$  in  $n_1$  and two queries  $?fluent(F), time(2), holdsAt(F, 2)$  and  $?fluent(F), time(2), not\ holdsAt(F, 2)$  in  $Q$ . These contribute just two answers  $holdsAt(available(lactose), 2)$  and  $holdsAt(available(glucose), 2)$  to  $R$ , which results in a Kernel Set  $K$  with one clause  $happens(use(lactose), 2) \leftarrow holdsAt(available(lactose), 2), holdsAt(available(glucose), 2)$ .

Each Kernel Set is generalised by the inductive phase  $P_3$ . Here, there is one abducible  $use/2$  in  $A_3$  and  $B$  is supplemented with five clauses in  $T_3$ : i.e.,  $happens(use(lactose), T) \leftarrow use(1, 0), try(1, 1, T), try(1, 2, T)$  in addition to  $try(1, 1, T) \leftarrow use(1, 1), holdsAt(available(lactose), T)$  and  $try(1, 1, T) \leftarrow not\ use(1, 1)$  as well as  $try(1, 2, T) \leftarrow use(1, 2), holdsAt(available(glucose), T)$  and  $try(1, 2, T) \leftarrow not\ use(1, 2)$ . But, there are no explanations  $U$  and hence no hypotheses  $H$  which can be obtained from this Kernel Set.

As a result, XHAIL revisits the abductive phase  $P_1$  to look for explanations  $\Delta$  with two atoms. One of these contains the facts  $happens(use(lactose), 2)$  and  $happens(use(glucose), 1)$ , which results in a corresponding Kernel Set  $K$  with the clause  $happens(use(glucose), 1) \leftarrow holdsAt(available(lactose), 1), holdsAt(available(glucose), 1)$  along with the clause  $happens(use(lactose), 2) \leftarrow holdsAt(available(lactose), 2), not\ holdsAt(available(glucose), 2)$ . Now, when this Kernel Set is generalised it yields the hypothesis  $H$  (by dropping the first literal of the first clause in the corresponding  $K'$ ).

A prototype implementation of XHAIL based on the Lparse/Smolens ASP solver took a couple of seconds to compute this hypothesis on a 1.66 GHz Centrino Duo laptop PC running Windows Vista with 1 GB of Ram.

## 6 Related Work

Several authors [4,19,7,25,28,5,51,13,39,1,43,49,56] have proposed methods for nonmonotonic ILP which are reviewed in [50]. Many use a generalisation of the stable model semantics, known as answer sets [16], for so-called extended logic programs with both NAF and classical negation. But, since extended programs are trivially reducible to normal programs via a simple renaming of classically negated literals [16], these approaches are no more general than the one presented in this paper. In fact, compared to XHAIL, existing methods impose strong restrictions on the learning task or they lack efficient strategies for guiding the computation. Some allow NAF in the hypothesis  $H$  but not in the theory  $B$ ; which means they not only lose much of the convenience and power of NAF but they also lose the ability to use the theory  $B \cup H$  as a subsequent starting theory. Others are restricted to learning stratified programs [3] or semi-normal default rules [47]. Most can only do Observation Predicate Learning (OPL) [37], where the predicates defined in  $H$  must appear in  $E$ . Initial experiments suggest that, in certain problems, XHAIL overcomes some of the limitations of these systems while effectively exploiting language and search bias. For example, in [2] XHAIL has been used to infer requirements from scenarios in a task that requires non-OPL learning of multiple predicates over non-stratified programs.

Sakama [50] defines a procedure for inducing extended programs under the skeptical answer set semantics. His method employs the notions of ‘relevance’ and ‘involvement’ to constrain the search space, but does not exploit language or search bias as effectively as XHAIL. Unlike XHAIL, which can generalise all the examples in one go, the procedure in [50] only considers one example at a time; which introduces a dependency on the order in which examples are presented. Moreover, it can only learn a single rule in response to each such example, is limited to OPL learning, assumes the example predicate appears nowhere in the theory, and imposes an additional restriction, called negative-cycle-freeness, on the hypothesis. However, all of these conditions are violated by the temporal modelling task in the previous section. It is mentioned in [50] that overcoming these constraints would necessitate the use of abductive reasoning — and indeed this is exactly what XHAIL achieves.

Otero [40] also considers the task of induction under the skeptical stable model semantics. He gives necessary and sufficient conditions for the existence of a stable model solution and describes a method for computing the corresponding models. But his method only returns ground unit hypotheses (i.e., the set of all ground atoms in a stable model). In fact, each solution returned by Otero’s procedure can formally be treated as a maximal abductive explanation of the examples. By contrast, XHAIL prefers minimal abductive solutions, which it subsequently generalises into non-ground inductive hypotheses.

Otero [41] also considers the task of learning temporal action theories from narratives describing a system’s dynamic behaviour. Like the case study in the previous section, his narratives comprise a set of known fluents and a set of known actions. But his theories are expressed in a formalism that combines various aspects of both the Event Calculus (EC) [24] and another formalism called the Situation Calculus (SC) [29]. A major concern of his approach is avoiding the so-called frame problem [53] which, in this context, refers to the need for many axioms describing the *non*-effects of actions on fluents. After showing how this problem is easily avoided by adding a few inertial axioms with NAF, the author notes how the limitations of earlier nonmonotonic ILP systems mean they cannot be used in this setting. Thus, Otero proposes a methodology by which the task of learning positive or negative action effects for a single fluent can be transformed into a monotonic learning problem. This transformation is based on distinguishing points at which a fluent is known to change (which become positive examples), from points at which fluent is known not to change (which become background knowledge), and from points at which it is unknown whether the fluent changes or not (which are denoted by Skolem symbols that are inserted as additional arguments into all action and fluent literals). He then suggests that the frame problem is avoided by excluding action effects whose conclusions are contained among their own preconditions and argues that his method is sound and complete for inducing the effects of actions without the frame problem.

However, the claims in [41] raise some issues. First, the frame problem, which results from failure of a representation formalism to provide a compact means of representing defaults and priorities, cannot be solved by further restricting any already inadequate language. The frame axioms that Otero disallows are not the cause of the problem, but merely a symptom; and excluding them only serves to compound the issue. But a rather more serious criticism of Otero’s approach concerns the restricted nature of the learning task it tackles. It is well known that the learning of action theories from partial observations cannot generally be solved by learning individual state transitions in isolation. For this reason it is easy to construct partial narratives containing just two fluents from which XHAIL can learn a correct specification of, say, a divide-by-two counter, but which Otero’s method cannot. Analogous limitations apply to the extended version of Otero’s procedure, described in [42], for learning the indirect effects of actions. Moreover, while Otero’s method is specifically designed to learn static laws and action effects or preconditions, XHAIL is a general purpose nonmonotonic ILP system which does not impose any a priori assumptions on the hypothesis space. Therefore, unlike Otero’s method, XHAIL can also be used — by suitably extending the language bias and background knowledge — to learn trigger axioms (as exemplified in the case study of previous section) in addition to more advanced constructs, such as release and trajectory axioms, or cumulative effects and cancellation laws, supported by alternative formulations of the Event Calculus [30,35].

Esposito et al. [12] describe a multi-strategy system called Inthelex which learns and revises normal programs from examples. It uses various operators to saturate, abstract, abduce, generalise and specialise clauses, respectively. Unlike XHAIL, which employs abduction to construct hypotheses whose head predicates may differ from those in the examples, Inthelex uses abduction only to hypothesise basic facts that might be missing from the description of each example. This means Inthelex is limited to the OPL learning task. In addition, Inthelex is restricted to the formalism of hierarchical linked datalog programs under the so-called object identity (OI) assumption [11], which is not appropriate for domains like the EC case study of the previous section. For, while any normal program can be transformed into linked datalog by flattening function symbols via the introduction of new predicates [48], this does not always preserve logical entailment [17]; and, while any (hierarchical) datalog program can be transformed into an equivalent datalog program under the OI assumption, this is potentially expensive [52]. Furthermore, the restriction to hierarchical programs means Inthelex is strictly less expressive than XHAIL. Although more details of its abductive methodology are given in [9] and [10] some key aspects concerning the Inthelex procedure are a little unclear: such as what happens when more than one abductive explanation is produced? Because the current Inthelex release does not support abstraction or abduction an independent evaluation of these features has not yet been possible.

There are two more differences between Inthelex and XHAIL. First, in the former approach, one example is processed at a time and is represented by a ground rule with a single head literal and zero or more body atoms. But this is equivalent, in the latter approach, to adding the body atoms of the selected example into the background theory. Second, in the Inthelex approach, clauses are revised by generalisation and specialisation operators. But these can be simulated, in the XHAIL approach, with some simple program transforms that exploit the nonmonotonicity of NAF. Indeed, the removal of literals from a clause can be achieved by the same technique used earlier in the inductive phase of the XHAIL procedure; except that *not del(I, J)* may be used in place of *use(I, J)* in order to minimise the number of literals deleted, as opposed to the number of literals used. Moreover, the addition of literals to a clause can be achieved by a variation of the method commonly used to learn the definition of an abnormality predicate newly inserted to the clause body. For example, a clause  $fly(X) \leftarrow bird(X)$  to be refined is first represented  $fly(X) \leftarrow bird(X), not\ ab(X)$  so that learning  $ab(X) \leftarrow penguin(X)$  gives the revised clause  $fly(X) \leftarrow bird(X), not\ penguin(X)$ . In general, the final clause is obtained by adding the complement of one body literal from each abnormality clause. If exactly one specialisation is required, then a restriction must be placed on the number of body literals in each abnormality clause (as zero means the clause will be deleted, and more than one means several clauses will be added). But more analysis of this technique is surely needed.

Tamaddoni-Nezhad et al. [55] describe how the non-OPL system Progol5 [37] can be applied to the prediction of enzyme inhibitions in biological networks. The authors propose a simple logical theory that uses NAF to model the effects of enzyme inhibition and non-inhibition on the concentration of compounds in a metabolic network. They go on to explain how possible inhibitions can be inferred from observed metabolite fluctuations by running the Progol5 system twice: the first time to generate a set of abducibles that explain the examples and the second time to generate a set of rules that generalise the abducibles. While the first invocation of Progol5 gives a similar result to the abductive phase of XHAIL, the latter overcomes several limitations of the former. In particular, Progol5 cannot abduce more than one atom in response to each example, is unable to abduce atoms that must be used more than once in a proof of the example, and has no way of reasoning abductively through programs with negation [44]. To compensate for this limitation, the authors must rewrite their logical model to exclude NAF before running Progol by adding an explicit truth value to some of the atoms. However, this rewriting is only possible because their initial model was a very simple stratified program.

Moyle and Muggleton [33] describe an application of Progol5 to the task of learning EC initiates and terminates axioms. But the limitations of Progol5 noted above necessitate a rewriting of the EC axioms to express the initiates and terminates axioms in terms of a single flips predicate with reified truth values. It also calls for ad-hoc transformations to ‘decouple’ the learning of initiating and terminating effects by creating artificial constraints to specify time intervals in which fluents are not clipped [32]. Moyle [31] describes an application of the ILP system Alecto to the learning of EC domain axioms from narratives. This system can be seen as a generalisation of Progol5 that uses a more powerful abductive procedure called SOLD resolution [58] to pre-process the examples. However SOLD resolution is also restricted to definite clauses and positive only examples. Thus, in order to learn EC theories, Alecto was extended by some as yet unspecified mechanism for handling negation. Since the current Alecto release (included in the latest Aleph distribution [54]) does not support abduction through negation an independent evaluation of these claims has not been possible.

Mueller [34] describes a method for rewriting a class of projection, planning, and model construction tasks in a Discrete Event Calculus (DEC) [35] to propositional satisfiability solving. This system cannot perform learning of DEC theories, while XHAIL can. Kakas and Riguzzi [23] describe a system for inductively learning abductive logic programs. But their system only performs OPL learning of target concepts that do not appear in the theory, and is only sound for the generalised partial stable model semantics, in which the truth of some literals may be undefined. Inoue [18] and Yamamoto [59] propose hypothesis finding systems which are sound and complete for full clausal logic. However, these systems only deal with monotonic classical negation.

## 7 Conclusions and Future Work

This paper introduced a general purpose nonmonotonic learning system, called eXtended Hybrid Abductive Inductive Learning (XHAIL), which exploits the full representation and reasoning power of Negation as Failure whilst using traditional forms of language and search bias to bound the search space. To do this, XHAIL integrates abductive, deductive, and inductive inference in a logical framework for the construction and generalisation of a preliminary ground Kernel Set. In this way, XHAIL allows for the non-observational multi-predicate learning of non-stratified programs. As an example of its utility, XHAIL was applied to the learning of temporal theories in a nonmonotonic Event Calculus. Since the task of modelling state and event based processes from observations is likely to gain importance, it is reasonable to suppose that systems like XHAIL will become increasingly useful in practical applications. However, the limitations of the proposed approach must be studied in greater detail and validated on some more realistic problems.

## Acknowledgments

The author is grateful to Dalal Alrajeh, Krysia Broda, Domenico Corapi and Alessandra Russo for useful discussions. Helpful comments were also provided by Peter Flach, Antonis Kakas and Ramon Otero. This work was supported by a Research Councils UK fellowship in Exabyte Informatics.

## References

- [1] H. Adé and M. Denecker. AILP: Abductive Inductive Logic Programming. In *14th International Joint Conference on Artificial Intelligence*, pages 1201–1207. Morgan Kaufmann, 1995.
- [2] D. Alrajeh, O. Ray, A. Russo, and S. Uchitel. Extracting Requirements from Scenarios with ILP. In *16th International Conference on Inductive Logic Programming*, volume 4455 of *LNCS*, pages 63–77. Springer, 2007.
- [3] K. Apt and R. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71, 1994.
- [4] M. Bain and S. Muggleton. Non-monotonic learning. In *Machine Intelligence 12*, pages 105–119. OUP, 1991.
- [5] F. Bergadano, D. Gunetti, M. Nicosia, and G. Ruffo. Learning logic programs with negation as failure. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 107–123. IOS Press, 1996.

- [6] K. Clark. Negation as failure rule. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.
- [7] Y. Dimopoulos and A. Kakas. Learning non-monotonic logic programs: Learning exceptions. In *8th European Conference on Machine Learning*, volume 912 of *LNAI*, pages 122–138. Springer, 1995.
- [8] K. Eshghi and R. Kowalski. Abduction compared with negation by failure. In *6th International Conference on Logic Programming*, pages 234–254. MIT Press, 1989.
- [9] F. Esposito, N. Fanizzi, S. Ferilli, T. Basile, and N. Di Mauro. Multistrategy operators for relational learning and their cooperation. *Fundamenta Informaticae*, 69(4):389–409, 2006.
- [10] F. Esposito, S. Ferilli, T. Basile, and N. Di Mauro. Inference of abduction theories for handling incompleteness in first-order learning. *Knowledge and Information Systems*, 11(2):217–242, 2007.
- [11] F. Esposito, A. Laterza, D. Malerba, and G. Semeraro. Refinement of Datalog programs. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming*, pages 73–94, 1996.
- [12] F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy Theory Revision: Induction and Abduction in INTHELEX. *Machine Learning*, 38(1/2):133–156, 2000.
- [13] L. Fogel and G. Zaverucha. Normal programs and multiple predicate learning. In *8th International Workshop on Inductive Logic Programming*, pages 175–184. Springer, 1998.
- [14] K. Furukawa. On the completion of the most specific hypothesis computation in inverse entailment for mutual recursion. In *1st International Conference on Discovery Science*, volume 1532 of *LNCS*, pages 315–325. Springer, 1998.
- [15] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *5th International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.
- [16] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.
- [17] K. Hirata. Flattening and implication. In *10th International Conference on Algorithmic Learning Theory*, volume 1720 of *LNAI*, pages 157–168. Springer, 1999.
- [18] K. Inoue. Induction as Consequence Finding. *Machine Learning*, 55(2):109–135, 2004.
- [19] K. Inoue and Y. Kudoh. Learning extended logic programs. In *15th International Joint Conference on Artificial Intelligence*, volume I, pages 176–181. Morgan Kaufmann, 1997.

- [20] F. Jacob and J. Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology*, 3:318–356, 1961.
- [21] A. Kakas, R. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.
- [22] A. Kakas and P. Mancarella. Generalized Stable Models: a Semantics for Abduction. In *9th European Conference on Artificial Intelligence*, pages 385–391. Pitman, 1990.
- [23] A. Kakas and F. Riguzzi. Abductive concept learning. *New Generation Computing*, 18(3):243–294, 2000.
- [24] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.
- [25] E. Lamma, F. Riguzzi, and L. Pereira. Strategies in combined learning via logic programs. *Machine Learning*, 38(1-2):63–87, 2000.
- [26] V. Lifschitz. Action languages, answer sets and planning. In K. Apt, V. Marek, M. Truszczynski, and D. Warren, editors, *The Logic Programming Paradigm: a 25 year perspective*, pages 357–373. Springer, 1999.
- [27] J. Lloyd. *Foundations of Logic Programming*. Springer, 1987.
- [28] L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 219–235. IOS Press, 1996.
- [29] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
- [30] R. Miller and M. Shanahan. Some alternative formulations of the event calculus. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, pages 452–490. Springer-Verlag, 2002.
- [31] S. Moyle. Using theory completion to learn a robot navigation control program. In *12th International Workshop on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 182–197. Springer, 2002.
- [32] S. Moyle. *An investigation into theory completion techniques in Inductive Logic Programming*. PhD thesis, University of Oxford, UK, 2003.
- [33] S. Moyle and S. Muggleton. Learning programs in the event calculus. In *7th International Workshop on Inductive Logic Programming*, volume 1297 of *LNAI*, pages 205–212. Springer, 1997.
- [34] E. Mueller. Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, 14(5):703–730, 2004.
- [35] E. Mueller. *Commonsense Reasoning*. Morgan Kaufmann, 2006.

- [36] S. Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 13(3-4):245–286, 1995.
- [37] S. Muggleton and C. Bryant. Theory Completion Using Inverse Entailment. In *10th International Conference on Inductive Logic Programming*, volume 1866 of *LNCS*, pages 130–146. Springer, 2000.
- [38] S. Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
- [39] P. Nicolas and B. Duval. Representation of incomplete knowledge by induction of default theories. In *6th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2173, pages 160–172. Springer, 2001.
- [40] R. Otero. Induction of Stable Models. In *11th International Conference on Inductive Logic Programming*, volume 2157 of *LNAI*, pages 193–205. Springer, 2001.
- [41] R. Otero. Induction of the effects of actions by monotonic methods. In *13th International Conference on Inductive Logic Programming*, volume 2835 of *LNCS*, pages 299–310. Springer, 2003.
- [42] R. Otero. Induction of the indirect effects of actions by monotonic methods. In *15th International Conference on Inductive Logic Programming*, volume 3625 of *LNCS*, pages 279–294. Springer, 2005.
- [43] L. De Raedt. *Interactive Theory Revision: An Inductive Logic Programming Approach*. Knowledge-based Systems. Academic Press, 1992.
- [44] O. Ray. *Hybrid Abductive-Inductive Learning*. PhD thesis, Department of Computing, Imperial College London, UK, 2005.
- [45] O. Ray. Using abduction for induction of normal logic programs. In *ECAI'06 Workshop on Abduction and Induction in Artificial Intelligence and Scientific Modelling*, pages 28–31, 2006.
- [46] O. Ray. Inferring process models from temporal data using abduction and induction. In *1st International Workshop on the Induction of Process Models*, 2007.
- [47] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.
- [48] C. Rouveirol. Flattening and saturation: Two representation changes for generalization. *Machine Learning*, 14(1):219–232, 1994.
- [49] C. Sakama. Some properties of inverse resolution in normal logic programs. In *9th International Workshop on Inductive Logic Programming*, volume 1634 of *LNCS*, pages 279–290. Springer, 1999.
- [50] C. Sakama. Induction from answer sets in nonmonotonic logic programs. *ACM Transactions on Computational Logic*, 6(2):203–231, 2005.

- [51] J. Seitzer. Stable ILP: exploring the added expressivity of negation in the background knowledge. In *IJCAI'95 Workshop on Frontiers of Inductive Logic Programming*, 1997.
- [52] G. Semeraro, F. Esposito, D. Malerba, N. Fanizzi, and S. Ferilli. A logic framework for the incremental inductive synthesis of datalog theories. In *7th International Workshop on Logic Programming Synthesis and Transformation*, volume 1463 of *LNCS*, pages 300–321. Springer, 1997.
- [53] M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.
- [54] A. Srinivasan. *The Aleph Manual (version 4)*. Machine Learning Group, Oxford University Computing Lab, 2003.
- [55] A. Tamaddoni-Nezhad, R. Chaleil, A. Kakas, M. Sternberg, J. Nicholson, and S. Muggleton. Modeling the effects of toxins in metabolic networks. *Engineering in Medicine and Biology Magazine*, 26(2):209–230, 2007.
- [56] K. Taylor. Inverse resolution of normal clauses. In *3rd International Workshop on Inductive Logic Programming*, pages 165–178. Joseph Stefan Institute, 1993.
- [57] A. Yamamoto. Which hypotheses can be found with Inverse Entailment? In *7th International Workshop on Inductive Logic Programming*, volume 1297 of *LNAI*, pages 296–308. Springer, 1997.
- [58] A. Yamamoto. Using Abduction for Induction based on Bottom Generalisation. In P. Flach and A. Kakas, editors, *Abduction and Induction: essays on their relation and integration*, volume 18 of *Applied Logic Series*, pages 267–280. Kluwer, 2000.
- [59] A. Yamamoto. Hypothesis finding based on upward refinement of residue hypotheses. *Theoretical Computer Science*, 298:5–19, 2003.