Mode-Directed Inverse Entailment for Full Clausal Theories

Oliver Ray[1] and Katsumi Inoue[2]

[1] University of Bristol, United Kingdom: oray@cs.bris.ac.uk
[2] National Institute of Informatics, Japan: ki@nii.ac.jp

Abstract. Mode declarations are an important form of language bias in ILP. But, while they are heavily utilised in Horn clause ILP, so far, they have not been applied to the more expressive full clausal setting. This paper presents a mode-directed ILP procedure for full clausal logic. It employs a first-order inference engine to abductively and inductively explain a set of examples with respect to a background theory. Each stage of hypothesis formation is guided by mode declarations using a generalisation of efficient Horn clause techniques for inverting entailment. Our approach exploits language bias more effectively than previous non-Horn ILP methods and avoids the need for interactive user assistance.

1 Introduction

Mode declarations [3] are an important form of language bias in ILP. They are a convenient way of constraining the hypothesis space searched by prominent ILP systems like Progol [3] and Aleph [8]. These systems use a type of Mode Directed Inverse Entailment (MDIE) [3] to construct and generalise a clause, called a Bottom Set [3], which bounds an otherwise intractable search space by acting as a syntactic and semantic 'bridge' between examples and hypotheses. But, while its practical utility has been convincingly shown in Horn clause applications of ILP, so far, no methods have been studied for lifting MDIE to the more expressive setting of full clausal logic in order to allow the representation and discovery of indefinite and disjunctive knowledge. The aim of our research is to show both the feasibility and the necessity of such an generalisation.

This paper presents a mode directed proof procedure for full clausal ILP. It uses a first-order inference engine called SOLAR [5] to implement a full clausal extension of an MDIE approach called HAIL [6]. HAIL combines abductive and inductive reasoning to construct and generalise a multi-clause variant of the Bottom Set called a Kernel Set [6]. SOLAR is an efficient tool for computing the deductive consequences of a given theory satisfying a sub-vocabulary known as a Production Field [1]. We use SOLAR to provide a full clausal realisation of HAIL, called fc-HAIL, by lifting the principles of MDIE from Horn clauses to the general case. By exploiting language bias more effectively than previous non-Horn ILP methods, fc-HAIL avoids the need for interactive user assistance. The utility of fc-HAIL is illustrated by a case study in fluid modelling.

The paper is structured as follows: Section 2 reviews the relevant background material; Sections 3 and 4 present our case study and the fc-HAIL approach; Section 5 compares fc-HAIL with related work; and Section 6 concludes.

2 Background

2.1 Notation and Terminology

This paper assumes a first-order language $\mathcal{L}$ with connectives $\wedge$, $\vee$, $\neg$, $\leftarrow$, $\rightarrow$, $\leftrightarrow$, logical constants $\top$, $\bot$, and a classical entailment relation $\models$. A literal L is either an atom A or its negation $\neg A$. The complement of L, denoted $\overline{L}$, is defined as A (resp. $\neg A$) if $L = \neg A$ (resp. A). A clause C is a disjunction of literals $L_1 \vee \ldots \vee L_m$ that will often be treated as the set of its disjuncts. Any variables are implicitly universally quantified at the front of the clause. A clause is Horn clause iff it has at most one positive literal and is full otherwise. A clause D subsumes C, written $D \geq C$, iff there exists a substitution θ such that $D\theta \subseteq C$. Moreover, D is said to properly subsume C iff $D \geq C$ but $C \not\geq D$. A theory T is a conjunction of clauses $C_1 \wedge \ldots \wedge C_n$ that will often be treated as the set of its conjuncts. A theory S subsumes T, written $S \geq T$, iff each clause in T is subsumed by at least one clause in S. We are concerned with the following ILP task [3]: Given theories B, E^+ and E^-, find a theory H such that $B \wedge H \models E^+$ and $B \wedge H \wedge E^- \not\models \bot$. We will require H to satisfy some syntactic constraints, specified by a set M of mode declarations, defined below.

2.2 Mode Declarations

Mode declarations are a well known form of language bias in ILP that are heavily exploited in this paper. As defined in [3], a mode declaration m is either a head declaration $modeh(r, l)$ or a body declaration $modeb(r, s)$ where r is an integer (the recall) and s is a ground literal (the scheme) which can contain the place-marker terms, $\#$, $+$ and $-$. Each scheme can be regarded as a 'template' with the place-markers standing for constants, input variables and output variables, respectively. The distinction between input and output variables is given by the restriction that any input variable in a body literal must be an input variable in the head or an output variable in some preceding body literal. In this way, a set M of mode declarations is associated with a set of clauses $\mathcal{L}_M$, called the language of M, such that $C = L_1 \vee L_2 \vee \ldots \vee L_n \in \mathcal{L}_M$ iff L_1 (resp. L_i for some $1 < i \leq n$) is obtained from some head (resp. body) declaration in M by replacing all $\#$ place-markers with constants and by replacing all $+$ (resp. $-$) place-markers with input (resp. output) variables. The schema of a mode declaration m is denoted $schema(m)$ and defined as the literal obtained from the scheme of m by replacing all place-markers by distinct variables. As explained in [3], the recall of a mode declaration is used to bound the number of times the scheme can be used.

Strictly, our definitions above differ from [3] in that we allow (positive or negative) literals in a scheme, while [3] only allows (positive) atoms. Though

mode declarations were conceived for Horn clauses, where a head literal is always positive and body literals are always negative, we note that the terminology *head* and *body* are still consistent with a full clausal viewpoint where head refers to the first literal L_1 in a clause C, and body refers to the remaining literals $L_2, \ldots, L_n$. Hence, our definitions naturally lift those in [3] from Horn clauses to full clauses. Of course, our definitions reduce to those of [3] in the Horn case using the equivalence $A \vee \neg B_1 \vee \ldots \vee \neg B_n \equiv A \leftarrow B_1 \wedge \ldots \wedge B_n$.

Example 1. If $M = \{ \quad modeh(1, uncle(+,+)), \quad modeb(1, \neg brother(+,-)), modeb(1, \neg father(+,+)) \ \}$ then $uncle(X,Y) \vee \neg brother(Y,Z) \vee \neg father(Z,Y) \in \mathcal{L}_M$ but $uncle(pete,Y) \vee \neg brother(Z,Y) \vee father(Z,Y) \notin \mathcal{L}_M$.

2.3 MDIE (Mode Directed Inverse Entailment)

MDIE [3] is a mode-directed ILP approach based on the semantics of Bottom Generalisation (BG) [3], which explains an example clause E relative to a background theory B by constructing and generalising an intermediate clause called a Bottom Set [3] of B and E. As explained in [3]:

- a Bottom Set of B and E is a ground clause $L_1 \vee \ldots \vee L_n$ such that $B \wedge \neg E\sigma \models \overline{L_i}$ for all $1 \leq i \leq n$ and for some Skolemising substitution σ
- a Skolem-free clause H is said to be computed by BG from B and E iff there exists a Bottom Set C of B and E such that $H \geq C$
- for any clause H computed by BG from B and E it holds that $B \wedge H \models E$

While BG is defined for full clauses, MDIE is restricted to Horn clauses. MDIE uses a mode-directed BG procedure within an ILP cover loop that selects seed examples E from a set E^+ of positive examples and ensures hypotheses H are consistent with a set E^- of negative examples. It uses a set M of mode declarations to heavily constrain the BG process. MDIE was first defined for Observation Predicate Learning (OPL) [4], but was later extended to the non-OPL setting using an approach called Theory Completion by Inverse Entailment (TCIE) [4]. This is a three phase method where, for each seed example E:

- a Bottom Set head literal is computed using a contrapositive reasoning method to find those head declaration instances that entail E
- some Bottom Set body atoms are computed using a query driven saturation process to find those body declaration instances that are entailed by $\neg E\sigma$
- a hypothesis is computed by a lattice based generalisation procedure using a top down search of the subsumption lattice bounded by the Bottom Set constructed above. A compression heuristic is used to search for consistent hypotheses that contain few literals and cover many positive examples.

2.4 HAIL (Hybrid Abductive Inductive Learning)

HAIL [6] is a mode-directed approach for realising the semantics of Kernel Set Subsumption (KSS) [7], which is based on a multi clause extension of the Bottom Set, called a Kernel Set [7]. As explained in [7]:

- a Kernel Set of B and E is a ground theory $K = \bigcup_{i=1}^{n} L_0^i \vee L_1^i \vee \ldots \vee L_{m_1}^i$ such that $B \wedge (L_0^1 \wedge \ldots \wedge L_0^n) \models E\sigma$ and $B \wedge L_j^i \models E\sigma$ for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$ and for some Skolemising substitution σ.
- a Skolem-free theory H is said to be computed by KSS from B and E iff there exists a Kernel Set K of B and E such that $H \geq K$
- for any theory H computed by KSS from B and E it holds that $B \wedge H \models E$

Example 2. If $B = \{p(X) \vee \neg q(X) \vee \neg q(Y)\}$ and $E = p(0,1) \vee \neg r$ then $H = \{q(X) \vee \neg r\}$ is derivable by KSS (but not BG) using $K = \{q(0) \vee \neg r\,,\, q(0) \vee \neg r\}$.

The Kernel Set plays an identical role to the Bottom Set by serving as an intermediate ground hypothesis that bounds the search space of an anti-subsumption generaliser. Moreover, HAIL is based on a three phase methodology which is directly analogous to MDIE in that:

- the contrapositive reasoning method of MDIE is replaced by a more general abductive procedure which can return solutions with more than one atom
- the query driven MDIE saturation process is applied, in turn, to each atom abduced in the previous step
- the MDIE compression-based search procedure is used to greedily generalise, in turn, each Kernel Set clause constructed above

By integrating abduction and induction in this way, HAIL overcomes some practical limitations of MDIE, such as an incompleteness of its contrapositive procedure and its inability to infer more than one clause for each example [**?**].

2.5 SOLAR (SOL resolution for Advanced Reasoning)

SOLAR is a proof procedure for clausal consequence finding [1]. It is based on Skip Ordered Linear (SOL) resolution [1], which extends connection tableaux [**?**] with a rule for 'skipping' literals. Skipped literals are assumptions made in proof and are needed for the completeness of SOL [1] for consequence finding. Since the closure of a theory may be infinite, SOLAR computes so-called characteristic [1] clauses, which represent 'interesting' theorems in some task. These are subsume minimal consequences that satisfy a sub-vocabulary specified by a form of language bias known as a production field [1]. A production field P is a pair $P = \langle Lits, Cond \rangle$ with a set of literals $Lits$ and a condition $Cond$. The language $\mathcal{L}_P$ of P is the set of clauses whose literals are instances of those in $Lits$ and which satisfy $Cond$. In practice, $Cond$ simply constrains certain properties of the clause such as its *length* and *depth*. If $Cond$ is empty, we write $P = \langle Lits \rangle$. If $Th(T)$ is the deductive closure of a theory T, and μT is the set of clauses in T not properly subsumed by another clause in T, then the characteristic clauses of T wrt P are defined as $Carc(T, P) = \mu(Th(T) \cap \mathcal{L}_P)$. If $\mathcal{L}_P$ is closed with respect to the inclusion of subsuming clauses, then P is said to be stable and SOLAR can compute clauses in $Carc(T, P)$ under efficient pruning strategies [5]. SOLAR's utility lies in the fact that many tasks, such as abduction and query answering can be reduced to the computation of characteristic clauses [1]. In this paper, we treat SOLAR as a black box. Given a theory T and production field P, we denote by $Solar(T, P)$ the set of consequences returned by SOLAR.

3 Motivating Example: Fluid Modelling

One benefit of non-Horn ILP is that it domain knowledge can be conveniently formalised in full first-order logic and automatically translated into clausal form. Such representational flexibility simplifies the knowledge engineering process by affording a more direct and less error prone formulation of prior knowledge. Another benefit is that non-Horn ILP enables the exploitation and discovery of disjunctive and indefinite concepts, which may be useful in formalising partial domain models. We illustrate these ideas in Example 3 below.

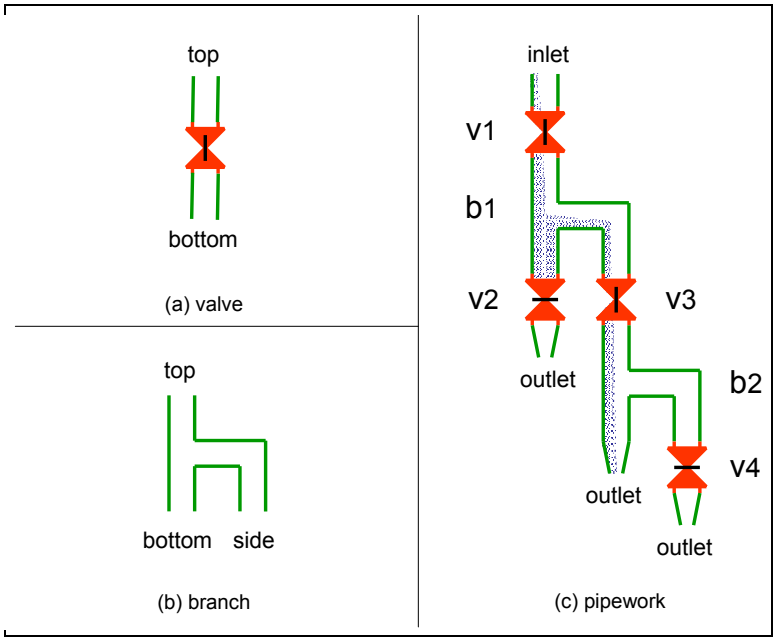

Fig. 1. Valves, branches, and pipework layout for Example 3

Example 3. This example concerns the flow of a liquid under gravity through a vertical system of pipework consisting of valves and branches. As illustrated in Figure 1, each valves has two points of connection, a top and a bottom, and it has a state which can be open or not open. Intuitively, water will flow from the top of the valve to its bottom if the valve is open (e.g. v1), but not otherwise (e.g. v2). By contrast, branches have three points of connection, a top, a bottom and a side. Ordinarily, water will flow from the top of a branch to its bottom (e.g. b2), but will be forced out of the side if it cannot flow out of the bottom (e.g. b1). A system of valves and branches is constructed by connecting the top of a branch or valve to the bottom or side of another branch or valve. Any unconnected points are designated inlets or outlets.

$$
\begin{aligned}
valve(X) &\rightarrow (\, source(bottom(X)) \leftrightarrow open(X) \wedge source(top(X)) \,) \\
valve(X) &\rightarrow (\, sink(top(X)) \leftrightarrow open(X) \wedge sink(bottom(X)) \,) \\
branch(X) &\rightarrow (\, source(bottom(X)) \leftrightarrow source(top(X)) \,) \\
branch(X) &\rightarrow (\, source(side(X)) \leftrightarrow source(top(X)) \wedge \neg sink(bottom(X)) \,) \\
branch(X) &\rightarrow (\, sink(top(X)) \leftrightarrow sink(bottom(X)) \vee sink(side(X)) \,) \\
branch(X) &\rightarrow (\, open(X) \,) \\
connect(X,Y) &\rightarrow (\, source(Y) \leftrightarrow source(X) \,) \\
connect(X,Y) &\rightarrow (\, sink(X) \leftrightarrow sink(Y) \,) \\
inlet(X) &\rightarrow (\, source(X) \,) \\
inlet(X) &\rightarrow (\, \neg connect(Y,X) \,) \\
outlet(X) &\rightarrow (\, sink(X) \,) \\
outlet(X) &\rightarrow (\, \neg connect(X,Y) \,)
\end{aligned}
$$

Fig. 2. Domain knowledge (D) for Example 3

$$
\begin{gathered}
branch(b1) \wedge branch(b2) \\
valve(v1) \wedge valve(v2) \wedge valve(v3) \wedge valve(v4) \\
inlet(top(v1)) \\
outlet(bottom(b2)) \wedge outlet(bottom(v2)) \wedge outlet(bottom(v4)) \\
connect(bottom(v1),\, top(b1)) \\
connect(bottom(b1),\, top(v2)) \\
connect(side(b1),\, top(v3)) \\
connect(bottom(v3),\, top(b2)) \\
connect(side(b2),\, top(v4))
\end{gathered}
$$

Fig. 3. Scenario description (S) for Example 3

To model the behaviour of a fluid flowing through a system of pipework, it is convenient to introduce the notions of source and sink. Intuitively, a source is a point into which fluid will flow from an inlet, while a sink is a point from which fluid would discharge into an outlet. [3] A formal model of these concepts is given by the domain theory D in Figure 2. It shows 12 first order implications (which are implicitly conjoined and universally quantified at the front). The first 2 implications formalise the behaviour of valves. They say the bottom of a valve is a source iff the top is a source and the valve is open; and vice versa. The next 4 formulae refer to branches: the bottom is source iff the top is a source; the side is a source iff the top is a source and the bottom is not a sink; the top is a sink iff the bottom or side is a sink; and a branch is always open. For any connected points X and Y, X is a source (sink) iff Y is a source (sink). Finally, an inlet (outlet) is a source (sink) which has no incoming (outgoing) connections.

[3] If a point is a source and a sink, then fluid is flowing through there (e.g. $side(b1)$). If it is a source but not a sink, then fluid is stagnating there (e.g. $bottom(b1)$).

$$B = D \cup S \qquad E^+ = \{source(bottom(b2))\} \qquad E^- = \{\neg open(v4)\}$$

$$M = \left\{ \begin{array}{l} modeh(*, open(+)) \\ modeh(*, \neg open(+)) \end{array} \right\} \cup \left\{ \begin{array}{l} modeb(*, outlet(bottom(+))) \\ modeb(*, \neg outlet(bottom(+))) \\ modeb(*, \neg valve(+)) \end{array} \right\}$$

$$H = \left\{ \begin{array}{l} open(X) \vee outlet(bottom(X)) \vee \neg valve(X). \\ \neg open(X) \vee \neg outlet(bottom(X)) \vee \neg valve(X). \end{array} \right\}$$

Fig. 4. Learning Inputs and Target Hypothesis for Example 3

The component and connection information for Figure 3 is given by the scenario description S in Figure 3, which is a conjunction of 15 ground atoms. Note how this description does not include the state of each valve as we wish to induce a hypothesis specifying precisely this. In particular we seek a hypothesis specifying which valves are open in order to explain the observation that fluid is flowing from the bottom of branch $b2$. If we suspect the valve state may depend upon whether its bottom is an outlet or not, and we require that $v4$ is not open, then we have the learning problem in Figure 4. Given this task, we would like to learn the hypothesis H stating that a valve is open iff its bottom is not an outlet: It is easy to check $H \in \mathcal{L}_M$ and a classical theorem prover can verify $B \wedge H \models E^+$ and $B \wedge H \wedge E^- \not\models \bot$. Thus H is a correct ILP hypothesis.

Transforming B to clausal form is a trivial operation that results in 21 Horn clauses and 2 non-Horn clauses, all with no more than 4 literals. One of the non-Horn clauses is $sink(bottom(X)) \vee sink(side(X)) \leftarrow sink(top(X)) \wedge branch(X)$. This clause is produced by the 5th formula in D, which models the preferential behaviour of a fluid to flow through the bottom of a branch instead of the side. It says the top of a branch is a sink only if the bottom or the side of the branch is a sink. This knowledge is disjunctive: if the top is a sink, then we know either the bottom is a sink or the side is a sink but, without further information, we may not know which. Given the non-Horn nature of B and H, a full clausal ILP approach is needed to solve this problem. Given the strong bias specified by M, this approach should utilise M in the learning process.

4 Full Clausal Hybrid Abductive Inductive Learning

In this section we show how SOLAR can be used to implement a full clausal generalisation of HAIL that efficiently solves the example above. To do this, we use SOLAR to lift all three phases of the HAIL learning cycle to full clausal logic by integrating the necessary reasoning tasks of abduction, query answering, coverage and consistency checking. We use $Solar(T, C)$ to denote the set of consequences returned by SOLAR for a theory T and production field P.

First we explain how SOLAR can be used for abduction in order to compute the head ;literals of a Kernel Set. This means computing a unit theory $\Delta = \{L_1, \ldots, L_m\}$ such that $B \wedge \Delta \models E\sigma$ and each L_i is an instance of a head declaration scheme in M (with place-markers replaced by variables). This is

equivalent to an abductive task: Given a theory T, a ground clause C (called a goal) and a set of literals Ls (whose instances are called abducibles), find a theory Δ (explanation) such that $T \wedge \Delta \models C$, each literal L_i is subsumed by a literal in Ls. We are interested in the subsume minimal explanations, as these lead to smaller Kernel Sets that are easier to generalise. Here, we do not explicitly require the consistency of Δ with T as this will be checked by fc-HAIL.

From previous work [1] and [5] it is known that SOLAR can compute such explanations by negating the characteristic clauses of the theory $T \wedge \neg C$ and production field $P = \{\overline{L} \,|\, L \in Ls\}$. This can be efficiently done when P is stable, by requiring that Ls is closed with respect to the inclusion of subsuming literals. But we wish to utilise head declaration schemas to constrain the function symbols in abduced literals, which usually violates stability:

Example 4. If $T = \{n(s(X)) \vee \neg n(X)\}$ and $C = n(s(s(s(0))))$ and $M = \{modeh(1, n(0))\}$ then the only stable production fields which can be used to compute the explanation $\Delta = \{n(0)\}$ by the abductive method in [1] must contain the literal $\neg n(X)$. But these also allow the alternative explanations $\Delta' = \{n(s(0))\}$ and $\Delta'' = \{n(s(s(0)))\}$ which are not allowed by M.

To avoid an inefficient generate-and-test approach, we developed a transformation for constraining terms using stable production fields that ensure the completeness of SOLAR's pruning strategies. I is based on the introduction of a new predicate p_L to represent each literal L in Ls. One clause expressing the relationship between p_L and L is added to T, along with the negation of C, and one maximally general instance of p_L is added to the production field. Any bindings computed by SOLAR are propagated back to the literals they represent. This is formalised in the *Abduce* procedure below, which is sound and complete for the computation of subsume minimal abductive explanations Moreover, this method leads to no loss of efficiency as it only requires one additional resolution step per abduced atom.

Example 5. If $T = \{n(s(X)) \vee \neg n(X)\}$ and $C = n(s(s(s(0))))$ and $Ls = \{n(0)\}$ then $Abduce(T, C, Ls) = \{n(0)\}$ using $B = \{n(s(X)) \vee \neg n(X), \neg n(s(s(s(0)))), p \vee n(0)\}$ and $P = \langle\{p\}\rangle$ where p is the proposition representing $n(0)$.

Procedure *Abduce*(T, C, Ls)
initialise $B = T \cup \{\overline{G} \,|\, G \in C\}$; $Ms = \emptyset$; $S = \emptyset$
for each literal $L \in Ls$ with variables $X_1, \ldots, X_n$\{
let p_L be a fresh predicate of arity n
let $B = B \cup \{L \vee p_L(X_1, \ldots, X_n)\}$
let $Ms = Ms \cup \{p_L(X_1, \ldots, X_n)\}$
}
for each clause $\Delta \in Solar(B, P)$ where $P = \langle Ms \rangle$) {
let $S = S \cup \{\{L\sigma \,|\, L \vee p_L(X_1, \ldots, X_n) \in B$ and $p_L(X_1, \ldots, X_n)\sigma \in \Delta\}\}$
}
return S

Our next task is to use SOLAR to implement the query answering engine used to compute the Kernel Set body literals. Given a theory T and a literal G, we wish to compute the instances of G that are entailed by T. From previous work [**?**], it is known SOLAR can efficiently compute such instances using the method of answer literals [**?**]. As formalised in the *Query* procedure below, this involves the use of a literal $ans(X_1, \ldots, X_n)$ to represent the variables $X_1, \ldots, X_n$ in G. The production field P is set to return any bindings to these literals (the length restriction avoids the computation of unnecessary disjunctive answers [9]). The bindings D are extracted from the literals computed by SOLAR and propagated back into the goal literal G to obtain the required instances A.

Procedure $Query(T, G)$
let $C = \overline{G} \vee ans(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are the variables in G
let $P = \langle \{ans(X_1, \ldots, X_n)\}, length \leq 1 \rangle$
let $S = Solar(T \cup \{C\}, P)$
let $D = \{\{X_1/t_1, \ldots, X_n/t_n\} \,|\, ans(t_1, \ldots, t_n) \in S\}$
let $A = \{G\sigma \,|\, \sigma \in D\}$
return A

The *Query* procedure is utilised in a full clausal generalisation of the Progol Bottom Set procedure[3], which uses a theory T and a set M of mode declarations to saturate a given head literal L with body literals up to some variable depth bounded by an integer d. As formalised in the *Saturate* procedure below, it maintains a growing set of input terms I that are used to replace + place-markers in the body declaration schemes. This results in a set of queries whose successful instances result in the insertion of new body literals to the clause X being constructed and the addition of new terms to I. Aside from the fact it uses a full clausal query engine, this exactly the procedure used by MDIE and Progol.

Procedure $Saturate(T, L, M)$
let $X = \{L\}$
let D be the first head declaration in M whose schema subsumes L
let I be the set of terms in L corresponding to $+$ place-markers in D
for each integer $i = 1, \ldots, d$ up to some depth d {
  for each body declaration C in M {
    let Q be the set of literals obtained from the scheme S of C by
    replacing all + place-markers by terms from I (in all possible ways)
    and replacing all other place-markers by distinct variables
    let $A = \bigcup_{q \in Q} Query(T, q)$
    add to X all literals in A
    add to I all terms in A corresponding to $-$ place-markers in C
}}
return X

All of the procedures are used in the fc-HAIL learning cycle, formalised below. Like Progol, it uses a covering loop to incrementally construct a hypothesis H using one uncovered example E at a time to focus the generalisation process. Here, we note that coverage and consistency checking can easily be performed by SOLAR. For example: a theory T is inconsistent iff $\emptyset \in Abduce(T, \emptyset, emptyset)$; and a theory T entails a clause C iff $T \wedge \neg C$ is inconsistent.

If it is non-ground, the seed example E is Skolemised by a substitution σ binding each variable to a fresh constant. A minimal abductive explanation Δ is then computed for the goal $E\sigma$ using the abducibles obtained from the head declaration schemas. Each atom of Delta is saturated with instances of the body declaration schemas to obtain a Kernel Set K. This is generalised to obtain a subsuming hypothesis H. [4]

Procedure $fc - HAIL(B, E^+, E^-, M)$
let $H = \emptyset$
let L be the set of head declaration schemas
while $B \cup H$ does not cover E^+ {
  let E be any seed example $E \in E^+$ not covered by $B \cup H$
  let σ be any Skolemising substitution for E
  let Δ be any abductive explanation $\Delta \in Abduce(B \cup H, E\sigma, L)$
    that is consistent with $B \wedge H \wedge E^+ \wedge E^-$
  let K be the Kernel Set $K = \bigcup_{L \in \Delta} Saturate(T \cup \{\overline{M\sigma}, \mid M \in E\}, L, M)$
  let H be any hypothesis such that $H \in \mathcal{L}_M$ and $H \geq K$
}
return H

Example 6. Consider B, E^+, E^- and M in Figure 4. Applying fc-HAIL, we have $L = \{\neg open(X), open(X)\}$. We must choose $E = source(bottom(b2))$. As E is ground we have $\sigma = \emptyset$. There is one minimal abductive explanation $\Delta = \{open(v1), \neg open(v2), open(v3)\}$ which results in the unique Kernel Set

$$K = \left\{ \begin{array}{l} open(v1) \vee outlet(bottom(v1)) \vee \neg valve(v1). \\ \neg open(v2) \vee \neg outlet(bottom(v2)) \vee \neg valve(v2). \\ open(v3) \vee outlet(bottom(v3)) \vee \neg valve(v3). \end{array} \right\}$$

As required, this is subsumed by the hypothesis

$$H = \left\{ \begin{array}{l} open(X) \vee outlet(bottom(X)) \vee \neg valve(X). \\ \neg open(X) \vee \neg outlet(bottom(X)) \vee \neg valve(X). \end{array} \right\}$$

Termination of fc-HAIL is ensured by bounding various parameters such as the maximum number of abducibles n in Δ. The worst case time complexity of HAIL is polynomial in n.

[4] We currently perform this generalisation by applying the Progol A^*-like search procedure, in turn, to each clause in K. This ensures that, in the Horn case, we always compute a hypothesis as or more compressive than Progol.

5 Related Work

Previous work on inverse entailment can be partitioned into two classes: (i) efficient and incomplete Horn clause systems, which use greedy covering loops and strong forms of language bias and search, such as Progol [3] and Aleph [8]; and inefficient but complete full clausal approaches, which use weaker forms of language and search bias, such as CF-Induction [2] and the Residue Procedure [10]. We see fc-HAIL as applying the efficient Horn techniques in (i) to the more expressive full clausal setting of (ii). We now use Example 3 to briefly compare fc-HAIL with these related approaches.

Progol and Aleph are direct Horn implementations of the MDIE described earlier. Given the inputs B, E^+, E^- and M in Figure 4, they cannot compute H as (i) B and H contain non-Horn clauses, (ii) it is necessary to infer two hypothesis clauses in order to explain a single example clause, and (iii) H does not entail any Bottom Set of B and E.

CF-Induction computes hypotheses by generalising a theory F obtained by negating a set CC of instances of characteristic clauses of $B \wedge \neg E$. In our example, the smallest possible theory F from which CF-Induction can infer the hypothesis H is shown below.

$$F = \left\{ \begin{array}{l} open(v1) \vee \neg outlet(bottom(v2)) \vee outlet(bottom(v3)) \vee \\ \quad outlet(bottom(v1)) \vee \neg valve(v3) \vee \neg valve(v2) \vee \neg valve(v1). \\ \\ \neg open(v2) \vee \neg outlet(bottom(v2)) \vee outlet(bottom(v3)) \vee \\ \quad outlet(bottom(v1)) \vee \neg valve(v3) \vee \neg valve(v2) \vee \neg valve(v1). \\ \\ open(v3) \vee \neg outlet(bottom(v2)) \vee outlet(bottom(v3)) \vee \\ \quad outlet(bottom(v1)) \vee \neg valve(v3) \vee \neg valve(v2) \vee \neg valve(v1). \end{array} \right\}$$

The theory F is much larger than the Kernel Set K and is therefore harder to generalise. Moreover, CF-Induction must work correspondingly harder than HAIL to construct F. In this case, CF-Induction computes a total of 32 characteristic clauses of which 7 must be selected for CC, while HAIL computes 1 explanation and executes 6 ground queries. This suggests that fc-HAIL is able to make more effective use of language bias than CF-Induction. By contrast, the Residue Procedure, which does not include any form of language bias, must generalise a Residue Hypothesis that is too large to be included in this paper.

6 Conclusions

This paper proposed a mode directed ILP procedure called fc-HAIL for full clausal logic. In particular, it used the SOLAR consequence finding engine to implement the HAIL learning cycle in a way that directly generalises efficient MDIE techniques from Horn clause logic to the more expressive general case. By avoiding syntactically disruptive and computationally expensive clausal form conversions, fc-HAIL exploits language bias more effectively than previous work on non-Horn ILP and removes the need for interactive user assistance. The

potential utility of fc-HAIL was illustrated with a case study modelling the flow of a fluid through a system of pipework. This example suggests that fc-HAIL's ability to automatically compute multi-clause non-Horn hypotheses in response to a single example may be useful in practice. We believe that non-Horn ILP will be useful in real applications to enable the representation and discovery of indefinite and disjunctive knowledge. We intend to test this claim using fc-HAIL by extending our model of fluid flow into a model of metabolic flux in biochemical networks. We also aim to better understand the trade-offs between expressivity and efficiency by investigating stronger forms bias and more powerful inference methods.

Acknowledgements

This work is supported by Research Councils UK (RCUK) and the Japan Society for the Promotion of Science (JSPS).

References

1. K. Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56(2-3):301–353, 1992.
2. K. Inoue. Induction as Consequence Finding. *Machine Learning*, 55(2):109–135, 2004.
3. S.H. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
4. S.H. Muggleton and C.H. Bryant. Theory Completion Using Inverse Entailment. In *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Computer Science*, pages 130–146. Springer Verlag, 2000.
5. H. Nabeshima, K. Iwanuma, and K. Inoue. SOLAR: A Consequence Finding System for Advanced Reasoning. In M.C. Mayer and F. Pirri, editors, *Proceedings of the 11th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 2796 of *Lecture Notes in Artificial Intelligence*, pages 257–263. Springer, 2003.
6. O. Ray, K. Broda, and A.M. Russo. Hybrid Abductive Inductive Learning: a Generalisation of Progol. In T. Horváth and A. Yamamoto, editors, *Proceedings of the 13th International Conference on Inductive Logic Programming*, volume 2835 of *Lecture Notes in Artificial Intelligence*, pages 311–328. Springer Verlag, 2003.
7. O. Ray, K. Broda, and A.M. Russo. Generalised Kernel Sets for Inverse Entailment. In B. Demoen and V. Lifschitz, editors, *Proceedings of the 20th International Conference on Logic Programming*, volume 3132 of *Lecture Notes in Computer Science*, pages 165–179. Springer Verlag, 2004.
8. A. Srinivasan. *The Aleph Manual (version 4)*, 2003. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/index.html.
9. M.E. Stickel. A Prolog technology theorem prover: A New Exposition and Implementation in Prolog. *Theoretical Computer Science*, 104:109–128, 1992.
10. A. Yamamoto. Hypothesis finding based on upward refinement of residue hypotheses. *Theoretical Computer Science*, 298:5–19, 2003.