Nimble: a hybrid abductive-inductive learning algorithm

Oliver Ray

October 2, 2002

Abstract

Learning is concerned with generalising from examples and improving with experience. Inductive Logic Programming (ILP) is a modern machine-based approach to learning that performs first-order theory completion from positive and negative examples. The most sophisticated and successful ILP system currently available is widely regarded as Progol5.

By considering the ILP problem in terms of a learning cycle that combines both *ab*ductive and *in*ductive reasoning, this paper proposes extensions to the theory and practice of Progol5.

Omega Ground Entailment ($\Omega_g E$) is introduced as an intuitive semantics that makes explicit one such connection between ILP and the learning cycle just mentioned. Nimble is introduced as an efficient proof system for $\Omega_g E$ that generalises Progol5 by integrating *ab*ductive and *in*ductive reasoning to provide a more complete instance of this cycle.

1 Introduction

Since its inception a decade ago Inductive Logic Programming (ILP) [Mug91] has made considerable strides in both theoretical and practical terms. Central to this success has been the development of efficient ILP systems and their application to real-world problems; and the current state-of-the-art in this respect is the Progol5 [MB00] system. By considering the ILP problem in terms of a learning cycle combining abductive and inductive reasoning, the aim of this paper is to extend the theory and practice of Progol5 and related systems.

Section 2 provides a detailed review of the successful and widely-applied Progol algorithm. Section 3 introduces the alternative semantics of Omega Ground Entailment ($\Omega_g E$) and reveals an *avoidable* source of incompleteness in Progol5. Section 4 introduces Nimble as an efficient hybrid abductive-inductive proof system for $\Omega_g E$ that subsumes and further generalises Progol5. All of these claims are illustrated with concrete examples: simple ILP problems that are solved by Nimble, but not by Progol.

2 Background

This section recalls the problem of Inductive Logic Programming (ILP) and motivates the approach of Inverse Entailment (IE). The Most Specific Hypothesis (MSH) approach to IE is introduced, and the efficient technique of Mode Directed Inverse Entailment (MDIE) is reviewed. The popular Progol4 algorithm is described and the important Progol5 extension is detailed. Finally the fundamental theoretical arguments that underpin these ideas are discussed.

2.1 Inductive Logic Programming (ILP)

Inductive logic programming (ILP) [Mug91] is an approach to inductive learning that emerged [MR94] §1 p629 at the intersection of Machine Learning (ML) and Logic Programming (LP). A familiarity with these fields is assumed, and the starting point of this paper is a widely accepted formulation of the ILP task [Mug95] §2 p2:

Definition (ILP). Given $\langle B, E, M\rangle \in \mathcal{L}_{Horn} \times \mathcal{L}_{Horn} \times \mathcal{L}_{Mode}$, Find $H \in \mathcal{L}_M$ such that

1. $B \wedge H \models E$
2. $B \wedge H \not\models \bot$
3. H has **Minimum Description Length**

Given **background knowledge** B and **examples** E, the task of ILP is to determine consistent **hypotheses** H that together with B entail E; where B, E and H are sets of Horn clauses and M is a set of domain-dependent **mode-declarations** that further constrain the hypothesis space $\mathcal{L}_M$ of H. Mode-declarations are discussed further in section 2.4, and for the purposes of this paper **minimum description length** means simply that H should contain as few literals as possible.

2.2 Inverse Entailment (IE)

One of the most modern and successful approaches to ILP, and the basis of the state-of-the-art Progol5 [MB00] algorithm, is **Inverse Entailment (IE)**. The methodology of IE is motivated by a simple and well-known classical equivalence which states that *inductive* hypotheses may be *deduced* in their negated form, from background knowledge together with the *negation* of the examples:

$$B \wedge H \models E \iff B \wedge \neg E \models \neg H \tag{1}$$

In reality the situation is just slightly more complicated than this, as it is usual in ILP to distinguish between **positive examples** E^+ and **negative examples** E^-, and between **knowledge** K and **integrity constraints** ic. And the convention [Mug95] §2 p2 is that negative clauses are considered to represent negative examples and integrity constraints, while positive clauses represent positive examples and background knowledge.

What is in fact required, is for B and H to **entail** all positive examples (called **coverage** in equation (2) below), but only to be **consistent** with all negative examples and integrity constraints (**integrity**). In this paper, as elsewhere, negative examples will often be treated as a special case of integrity constraint, and it will be assumed that in case there are no integrity constraints, then B and H are consistent with each other. More accurately the criteria for H become:

$$\begin{array}{rl} B \wedge \neg E^+ \models \neg H & \textit{coverage} \\ B \wedge ic \not\models \neg H & \textit{integrity} \end{array} \tag{2}$$

Example 2.1 (Fast Food). The following example illustrates the use of positive and negative examples. B, H and E are expressed as sets of Horn clauses, where each clause is of the form *head* $\leftarrow$ *body*. The usual conventions apply so that for example $\mathit{offer}(\mathit{mcDonalds})$ represents a **fact** (a clause with an empty body), and $\bot \leftarrow \mathit{meal}(\mathit{theRitz})$ represents a *negative* example (a clause with an empty head denoted '$\bot$'), 'meal' is a **predicate** symbol, and 'R' is a **variable** symbol. As usual, clauses are implicitly universally quantified and conjoined, and where necessary terminated by full stops.

$$\begin{array}{rcl} B & = & \left\{ \begin{array}{l} \mathit{meal}(R) \leftarrow \mathit{getBurger}(R),\ \mathit{getFries}(R). \\ \mathit{getBurger}(R) \leftarrow \mathit{getFries}(R),\ \mathit{offer}(R). \\ \mathit{getBurger}(\mathit{theRitz}). \\ \mathit{offer}(\mathit{mcDonalds}).\ \mathit{offer}(\mathit{burgerKing}).\ \mathit{offer}(\mathit{wimpy}). \end{array} \right. \\ E & = & \left\{ \begin{array}{l} \mathit{meal}(\mathit{mcDonalds}).\ \mathit{meal}(\mathit{burgerKing}).\ \mathit{meal}(\mathit{wimpy}). \\ \bot \leftarrow \mathit{meal}(\mathit{theRitz}). \end{array} \right. \\ H & = & \left\{ \mathit{getFries}(R) \leftarrow \mathit{offer}(R). \right. \end{array}$$

The background knowledge states that to say one has had a meal in a restaurant R, it is enough that one gets both a burger and french-fries there. One automatically gets a burger if one gets fries at a restaurant participating in a special offer. It is known that such an offer is available in the restaurants indicated, and that one has partaken of a burger in the Ritz.

There are three positive examples, which upon negating and abbreviating constant names in the obvious way yield $\neg E^+ = \bot \twoheadleftarrow meal(md),\ meal(bk),\ meal(w)$. There is one negative example $E^- = \bot \twoheadleftarrow meal(r)$. There are no other integrity constraints. It is easily verified that H is a valid inductive hypothesis according to equation (2):

Proof. **Coverage**: $B \wedge \neg E^+ \models \neg H$. For assume the contrary, then $B \wedge \neg E^+ \not\models \neg H$ so that $B \wedge \neg E^+ \wedge H \not\models \bot$ which is a certainly a contradiction as the body atoms of the negative clause $\neg E^+$ are all derivable from B and H. ∎

Proof. **Integrity**: $B \wedge E^- \not\models \neg H$. For assume the contrary, then $B \wedge E^- \models \neg H$ so that $B \wedge E^- \wedge H \models \bot$ which is a certainly a contradiction as the body atom of the only negative clause E^- is not derivable from B and H. ∎

2.3 Most Specific Hypothesis (MSH)

It is one thing to verify the validity of a given hypothesis, and another to efficiently construct such a hypothesis given only the background knowledge and examples. Efficient realisations of IE make use of a key construction referred to as the **Most Specific Hypothesis (MSH)**[1] [Mug95] Def24 p17, or **Bottom Set** (BS) [Yam96] Eq1 p2, which is defined as the disjunction of all ground literals whose negation may be deduced from B together with the negation of *a single* example e. A hypotheses h is then found by generalising this construction, which using the notation of [Yam96] will henceforth be denoted $Bot(B, e)$.

$$h \models Bot(B,e) \Longrightarrow B \wedge h \models e \tag{3}$$

where

$$Bot(B,e) \ := \ \bigvee \{ l \in \text{GndLits} \mid B \wedge \neg e \models \neg l \} \tag{4}$$

Example 2.2 (Fast Food: revisited). Consider again the problem of example 2.1 and assume that the first positive example is chosen as the seed. In this case:

$$\begin{array}{rcl} B & = & \left\{ \begin{array}{l} meal(R) \twoheadleftarrow getBurger(R),\ getFries(R). \\ getBurger(R) \twoheadleftarrow getFries(R),\ offer(R). \\ getBurger(r). \\ offer(md).\ offer(bk).\ offer(w). \end{array} \right. \\ e & = & \{\ meal(md). \\ h & = & \{\ getFries(R) \twoheadleftarrow offer(R). \end{array}$$

Given B and e as indicated above, $Bot(B, e) = meal(md) \vee getFries(md) \vee \neg getBurger(r) \vee \neg offer(md) \vee \neg offer(bk) \vee \neg offer(w)$. At the same time, by universal instantiation $h \models getFries(md) \twoheadleftarrow offer(md)$ which is logically equivalent to $getfries(md) \vee \neg offer(md)$. And so $h \models Bot(B, e)$ by $\vee$-introduction. It is also of interest that in this case we have the stronger condition that $h \preccurlyeq Bot(B, e)$, where $\preccurlyeq$ denotes θ-subsumption.

[1]The *Bottom Set* concept was first introduced in [Mug95] Def24 p17 with the terminology '*Most Specific Clause*' and notation '$\bot$'. These same conventions are also employed in [MF01]. In order to avoid confusion with the logical symbol for falsity, the notation '$Bot(B, e)$' was introduced in [Yam96] Eq1 p2, along with the terminology '*Bottom Set*'. The notation '$\bot(B, E)$' is used in [MB00] and the terminology '*Most Specific Hypothesis*' is used in [FO00].

2.4 Mode Directed Inverse Entailment (MDIE)

Although MSH has a particularly simple formalisation, it non-trivial to realise general mechanisms for computing efficiently within that formalism. Two immediate obstacles: i) in general $Bot(B, e)$ may be *infinite* [Mug95] §8 p16, and ii) there is *no* effective procedure for constructing *all* possible h. Mode Directed Inverse Entailment (MDIE) [Mug95] is currently the most successful means of addressing these problems. MDIE approaches make considerable use of user-specified **mode-declarations** [Mug95] Def20 p16 to construct a finite resource-bounded subset of $Bot(B, e)$, and then to perform an efficient θ-subsumption search for the best h.

Mode-declarations play a fundamental role in MDIE by providing a language in which to specify syntactic restrictions on the induced hypotheses. Mode-declarations come in two varieties: head-declarations, which impose restrictions on atoms in the heads of induced clauses, and body-declarations which apply to body atoms. Mode-declarations not only determine outright the **predicates** that may appear in head and body atoms, but through the notion of **variable type** also impose a quasi-ordering[2] on the occurances of variables in induced clauses. And finally the notion of **recall** is used to bound the number of solutions investigated in SLD computations.

Definition (Mode-Declaration). [3] A **placemaker** is one of three things: +type, -type or #type. A **scheme** is any ground atom with placemarkers optionally appearing in place of constants. A **mode-declaration** has one of two forms: a **head**-declaration modeh(n,s) or a **body**-declaration modeb(n,s) where n is a positive integer called the **recall** and s is a scheme. Given a set M of mode-declarations, M_h and M_b will represent respectively the set of head-declarations and body-declarations in M. If m is a mode-declaration then **recall(m)** will represent the recall n of m, and **atom(m)** will represent the scheme s of m with every placemarker occurance replaced by a *distinct* variable.

Example 2.3. Both $m_1 = modeh(1, getFries(+restaurant))$ and $m_2 = modeb(1, offer(+restaurant))$ are valid mode-declarations and the hypothesis $getFries(R) \leftarrow offer(R)$ falls within the language defined by these declarations (where R is assumed to be of type restaurant). In addition $recall(m_1) = 1$ and $atom(m_2) = offer(X)$.

Notation ($\overline{X}$). In order to compact somewhat the following presentation, the notation $\overline{X}$ will often be used in place of $\neg X$ to denote negation.

2.4.1 BottomSet

The algorithm called BottomSet below is *essentially* that used by (all) Progols to construct their approximation of $Bot(B, e)$. The presentation below is *based* on [Mug95] Alg40 p40 and [MF01] §4.5 but corrects certain ommissions in the handling of depth[4] and head-declarations[5], and clarifies certain ambiguities[6] in the original presentations. BottomSet takes as input four things: i) a set of body-declarations M_b, ii) a definite logic program B representing background knowledge, iii) a definite clause e representing an example, iv) a head-declaration h.

In order to compute with $\overline{e}$ as required by equation (4), the first step is to add the skolemised body atoms e^- to B while continuing to work with the Skolemised head e^+. (This process is discussed in more detail in section 3.2). BottomSet assumes a function $hash(t)$ that maps each term to a *unique* natural number. Strictly speaking, BottomSet does not return the raw $Bot(B, e)$, which is defined as a disjunction of *ground* literals, but instead it introduces variables where so required by the language bias M. The system parameter N_d is referred to as the **maximum depth** [Mug95] Def10 p11.

[2] input variables in body atoms must occur either as input variables in the head atom or else as output variables in *preceding* body atoms. Output variables in the head atom must occur either as inputs in the head or else as outputs in the body

[3] based on Muggleton [Mug95] Def20 p16

[4] variables of depth $n + 1$ should only be introduced after all atoms of depth n have been constructed

[5] Progol investigates all head-declarations, not just the one as suggested in the original presentation

[6] for example e^+ should not be added to B along with e^-

The notation $P \vdash_{sld} g\theta$ means that query $?g$ succeeds from program P under SLD-resolution with answer substitution θ. In Progol this is determined by an in-built Prolog interpretor. It is further assumed that any computation performed by the SLD interpreter will be failed after at most N_h resolution steps, where N_h is a system parameter referred to as the **resolution bound**. It should be noted that the internal interpreter *supports* many of the standard in-built predicates and *permits* the use of negation-as-failure (although this lies outside the scope of the sematics).

Algorithm 2.4 (BottomSet). [7]

```
0. input        given $M_b$, $B$, $e$, $h$
1. initialise   add $e^-$ to $B$ and replace $e$ by $e^+$
                set $InTerms = BS = \emptyset$
                let $g = atom(h)$
                if $g \preccurlyeq_\theta e$ then
2. construct        for each replacement $v/t$ in $\theta$
   head                 if $v$ is #type then
                            replace $v$ in $g$ by $t$
                        else
                            replace $v$ in $g$ by $v_{hash(t)}$
                        if $v$ is +type then
                            add $t$ to $InTerms$
                        add $g$ to $BS$
3. construct        repeat $N_d$ times
   body                 initialise $NextTerms = \emptyset$
                        for each body-declaration $b$ in $M_b$
                            let $a = atom(b)$
                            for each substitution $\theta'$ of +variables in $a$ by $InTerms$
                                repeat $recall(b)$ times
                                    for each substitution $\theta''$ s.t. $B \vdash_{sld} a\theta'\theta''$
                                        for each replacement $v/t$ in $\theta'\theta''$
                                            if $v$ is #type then
                                                replace $v$ in $a$ by $t$
                                            else
                                                replace $v$ in $a$ by $v_{hash(t)}$
                                            if $v$ is -type then
                                                add $t$ to $NextTerms$
                                        add $\overline{a}$ to $BS$
                        add $NextTerms$ to $InTerms$
4. output       return $BS$
```

2.4.2 Progol4

Progol [Mug95] was the first algorithm implement the principles of MDIE, and remains today a state-of-the-art and extremely popular, widely applied and highly successful system. In common with other ILP systems, the aim of Progol is to find an H that together with B entails E. The Progol language bias is as follows. B is represented as a set of Horn clauses with negative clauses being interpreted as integrity constraints and treated internally as having the reserved atom $false$ in thier head. E is also represented as a set of Horn clauses with negative clauses standing for negative examples. H is a set of Horn clauses: i) consistent with any integrity constraints, and ii) expressed within the language defined by a set of mode-declarations M.

Progol uses a greedy cover set approach to construct hypothesis clauses one-by-one. This means that while there are outstanding examples, one of them e_i is chosen as a seed, and a search

[7] based on Muggleton [Mug95] Alg40 p40 and [MF01] §4.5

is undertaken for a suitable clause h_i that covers this example and achieves maximal compression with respect to the remaining examples E. The h is then asserted and any covered examples are retracted.

Early Progols (prior to version 5), in common with most other ILP systems, impose one significant restriction on h_i: namely that the predicate in the head of h_i must the same predicate in the head of the seed example e_i. This important restriction is referred to as **observation predicate learning (OPL)** [MB00], and is largely a vestage of early machine learning systems. We will presently see how Progol5 is able to overcome this restriction by cleverly extending the core Progol algorithm, but first the review of Progol4 is concluded.

The hypothesis h_i is obtained in two steps. In the **saturation** step the most specific clause (BottomSet) is constructed (within the specified language bias and resource bounds) that entails e_i. In the **reduction** step a general to specific Search is performed over the subsumption lattice bounded below by the BottomSet, and above by the empty clause. The search criteria takes into account integrity constraints, the number of positive and negative examples covered, and an estimate of the total number of atoms required in the clause[8]].

The strategy used by Progol to search the bounded θ-subsumption lattice is described in [Mug95] §D.2 p41 and will not be discussed further here. The presentation below of the core Progol cover set algorithm is based on Muggleton [Mug95] Alg44 p43 and [MF01] §4.8, but is closer to the Progol4 implementation in its use of head-declarations[9].

Algorithm 2.5 (Progol4). [10]

```
0. input            given $M$, $B$, $E$
1. cover set        while $E$ not empty
2. seed                 choose seed example $e_i$ from $E$
3. head-mode            for each head-declaration $h_j$ in $M_h$ s.t. $atom(h_j) \preccurlyeq e_i$
4. saturation               construct $BS_{ij} = BottomSet(M_b, B, e_i, h_j)$
5. reduction                find $C_{ij} = Search(BS_{ij}, E)$
6. hypothesis           let $h_i$ be the most compressive $C_{ij}$ over all $j$
                        add $h_i$ to $B$
7. cover removal        remove covered examples from $E$
8. output           return
```

2.5 Theory completion using Inverse Entailment (TCIE)

In many real-world applications the limitations of OPL considerably restrict the applicability of ILP techniques. Theory completion using Inverse Entailment (TCIE) [MB00] is an approach that addresses these limitations within the methodology of MDIE. The motivation behind TCIE is the observation that the algorithm BottomSet only computes negative ground literals in $Bot(B, e)$ (or equivalently the *positive* ground unit consequences of $B \wedge \overline{e}$). TCIE augments the BottomSet construction with a mechanism (called StartSet below) for computing the *negative* ground unit consequences of $B \wedge \overline{e}$. The atoms in StartSet are candidates for the head of the hypothesis, just as those in BottomSet are candidates for the body.

The approach adopted by TCIE is based on Stickle's Prolog Technology Theorem Prover (PTTP) and in particular the technique of contra-positive locking. This essentially involves rewriting each n-atom definite clause in n logically equivalent indefinite forms and introducing new predicates in place of negated atoms, to leave n definite clauses. For example the clause $a \twoheadleftarrow b, c$ would be represented as $a \twoheadleftarrow b, c$ and $b^* \twoheadleftarrow a^*, c$ and $c^* \twoheadleftarrow a^*, b$; where a^*, b^* and c^* are newly introduced predicates to *represent* the negations of a, b and c. In effect, this technique allows negative information to be propagated backwards through the original rules. For example, given b but not a, under locking it can be correctly concluded by SLD resolution that not b.

[8]hence it is described as an 'A*-like search

[9]see footnote 5

[10]based on Muggleton [Mug95] Alg44 p43 and [MF01] §4.8

2.5.1 StartSet

The StartSet algorithm operates by adding temporarily to the knowledge base contra-positives of B and the negative form of e. Members of StartSet are decided by those atoms g which succeed as queries in their negative form $?g^*$. Since all rules with a negative atom in their head have a negative atom in their body, and since the only negative fact is provided by e, contra-positives need only be formed for those rules whose head predicate occurs in some path in the call-graph of B that starts from the predicate in the head of e and ends on query predicate. In order to avoid a proliferation of contra-positives, the algorithm considers just one head predicate p at a time, and therefore requires only those head-declarations $M_h(p)$ defined for that predicate. For convenience, the algorithm *below* returns viable head atoms *together* with their corresponding head-declarations.

Algorithm 2.6 (StartSet). [11]

```
0. input          given $M_h(p)$, $B$, $e$, $p$
1. initialise     add $e^-$ to $B$, replace $e$ by $e^+$ and set $SS = \emptyset$
2. locking        $B = B \cup \{e^*\} \cup \{b_j^* \twoheadleftarrow b_0^*, \ldots, b_{j-1}, b_{j+1}, \ldots, b_m \mid b_0 \twoheadleftarrow b_1, \ldots, b_j, \ldots, b_m \in B\}$
                   where $m, j > 0$ and the prediate in $b_0$ occurs in some path in the call-graph
                   of $B$ starting from the predicate in $e$ and ending on $p$
3. construct      for each head-declaration $h$ in $M_h(p)$
   SS                 let $g = atom(h)$
                      for each substitution $\theta$ s.t. $B \vdash_{sld} g^*\theta$
                          add $\langle g\theta, h\rangle$ to SS
4. output         return $SS$
```

2.5.2 Progol5

Progol5 [Mug95] is the latest and most powerful member of the Progol family, and is the first algorithm to implement the principles of TCIE. Progol5 employs the same cover set approach as Progol4 and begins by picking a seed example e to generalise. Each possible head predicate p_i is then tried in turn, and a StartSet SS_i is constructed for that example-predicate combination. Each possible head atom s_{ij} returned is then tried with its corresponding head-declaration. In order to reuse as-is the BottomSet algorithm, the selected head atom is first grafted onto the body of the example. The Search algorithm is the same as used by Progol4. Once all possible head atoms have been tried, the most compressive clause overall is added to B and returned.

Algorithm 2.7 (Progol5). [12]

```
0. input              given $M$, $B$, $E$
1. cover set          while $E$ not empty
2. seed                   choose seed example $e$ from $E$
3. head predicate         for each predicate $p_i$ mentioned in $M_h$
4. contra-positives           construct $SS_i = StartSet(M_h(p_i), B, e, p_i)$
5. head atom                  for each atom $s_{ij}$ in $SS_i$ with corresponding head declaration $h$
6. transform seed                 form $e'_{ij} = s_{ij} \twoheadleftarrow e^-$
7. saturation                     construct $BS_{ij} = BottomSet(M_b, B, e_{ij}, h)$
8. reduction                      find $C_{ij} = Search(BS_{ij}, E)$
9. hypothesis             let $C$ be the most compressive clause over all
                          add $C$ to $B$
10. cover removal         remove covered examples from $E$
11. output            return
```

[11]based on Muggleton [MB00] §2.1
[12]based on Muggleton [MB00] §2.3

2.6 Theoretical Foundations

All of the methods considered so far in this section are refinements of the MSH generalisation approach and as such they all rest on the claim repeated below from §2.3.

$$h \models Bot(B,e) \Longrightarrow B \wedge h \models e \tag{5}$$

where

$$Bot(B,e) \ := \ \bigvee\{l \in \text{GndLits} \mid B \wedge \overline{e} \models \overline{l}\} \tag{6}$$

And the reasoning that underpins this claim is first introduced in [Mug95] §7 p15 and then elaborated upon in [MF01] §4.1. For convenience this reasoning is repeated below in its latter form. The argument is presented verbatim, *except* that the notation $\perp$ has been replaced by $Bot(B,e)$ and one footnote and one pair of brackets have been inserted.

> Given background knowledge B and examples E find the simplest consistent hypothesis H such that
>
> $$B \wedge H \models E \tag{7}$$
>
> If we rearrange the above using the law of contraposition we get the more suitable form
>
> $$B \wedge \overline{E} \models \overline{H} \tag{8}$$
>
> If we restrict H and E to being single Horn clauses,
>
> $$\overline{H} \text{ (and } \overline{E}\text{) above will be ground Skolemised unit clauses} \tag{9}$$
>
> If $\overline{Bot}(B,E)$ is the conjunction of ground literals which are true in all models[13] of $B \wedge \overline{E}$ we have
>
> $$B \wedge \overline{E} \models \overline{Bot}(B,E) \tag{10}$$
>
> Since $\overline{H}$ must be true in every model of $B \wedge \overline{E}$ it must contain a subset of the ground literals in $\overline{Bot}(B,E)$. Hence
>
> $$B \wedge \overline{E} \models \overline{Bot}(B,E) \models \overline{H} \tag{11}$$
>
> and so
>
> $$H \models Bot(B,E) \tag{12}$$
>
> A subset of the solutions for H can then be found by considering a those clauses which θ-subsume $Bot(B,E)$.

[13] i.e. the conjunction of *all* ground literals entailed by B and $\overline{E}$

3 Omega Ground Entailment ($\Omega_g E$)

This section introduces Omega Ground Entailment[14] ($\Omega_g E$) as an alternative semantics to MSH generalisation. There are two motivations for doing this: i) to clarify a number of subtleties that will emerge from a detailed examination of existing arguments, ii) to provide a convenient framework in which to analyse and generalise existing systems. This section will argue:

- $\Omega_g E$ is a highly intuitive semantics that does not involve negation
- $\Omega_g E$ is provably equivalent to MSH generalisation: the cannonical semantics of Progol5
- $\Omega_g E$ enforces a clear separation of abductive, deductive and inductive reasoning components
- $\Omega_g E$ is a particularly convenient framework for the analysis of Progol5
- $\Omega_g E$ suggests an immediate and effective extension to Progol5

The structure of this section is as follows: Section 3.1 performs a detailed analysis of existing MSH arguments and reveals some subtle issues relating to Skolemisation. Section 3.2 address these issues by explicitly incorporating Skolemisation into the semantics. Section 3.3 motivates the Omega Ground Clause $\Omega_g E$ as an alternative to the MSH $Bot(B, e)$, and demonstrates their equivalence.

Section 3.4 performs a detailed analysis of Progol5's StartSet algorithm with respect to $\Omega_g E$, and reveals a surprising source of incompleteness. Two simple strategies for removing this incompleteness are presented, and that Progol5 does indeed suffer from this incompleteness is demonstrated with a simple example input file. Section 3.5 introduces a hybrid abductive-deductive learning cycle of which Progol is itself a *partial* instance.

The incompleteness mentioned above will be seen to be isolated in the abductive phase of Progol5. Considerations based on the learning cycle as a *whole* will be used in later sections to achieve still further generalisation over existing algorithms. In particular the Nimble algorithm will be motivated in section 4 as further strengthening both the abductive and inductive phases of the cycle.

3.1 Motivaton

The argument presented in section 2.6 above, is not entirely unproblematic. As pointed out by Yamamoto, it is clearly intended to demonstrate: i) the *equivalence* $B, H \models E \iff H \models Bot(B, E)$, and ii) only a subset of the H may be found using θ-subsumption. That the intended equivalence is false was first remarked by [Yam96] §1 p2, who went on to show [Yam97] that completeness does in fact hold for Plotkin's relative subsumption, but not for entailment in general[15]. Yet it seems, perhaps because the incompleteness reported by Yamamoto was somehow attributed to the use of θ-subsumption in the search for H, that no detailed examination of the original argument was carried out. This is unfortunate as the argument is in fact flawed.

In the **forward direction** the best way to see the error is to consider equation (11) and the reasoning that immediately precedes it. The argument rests on two premises: i) that $B \wedge \overline{E} \models \overline{H}$ (8), and that ii) $\overline{H}$ is *a* set (conjunction) of ground literals (9). The argument then runs that since $\overline{Bot}(B, E)$ is *the* set (conjunction) of *all* ground literals entailed by $B \wedge \overline{E}$, therefore H must contain a subset of these - or else it could not possibly be itself entailed by $B \wedge \overline{E}$.

While this argument is sound, the premises are invalid: specifically, if H is non-ground then one or other will be falsified. Either $\overline{H}$ will be an existentially *quantified* conjunction of non-ground *non*-Skolemised literals and (9) will be falsified, or else $\overline{H}$ must be Skolemised - in which case (8) will be falsified. This is because after Skolemisation $\overline{H}$ will contain formulae with Skolem

[14]It is stated without comment that $\Omega_g E$ is a special case of a more general semantics that will not be discussed further in this paper.

[15]see also [Mug98]

constants that by definition do not even appear in $B \wedge \overline{E}$, and which need not therefore be entailed by $B \wedge \overline{E}$. For consider a trivial example:

$$\begin{aligned} B &= \emptyset \\ E &= p(X) \\ H &= p(X) \end{aligned}$$

Clearly $B \wedge H \models E$. But either we accept that $\overline{H} = \exists X[\neg p(X)]$ in which case it is not a conjunction of ground literals. Or else we Skolemise and $\overline{H} = \neg p(a)$, which is not entailed by $B \wedge \overline{E}$. This is true whether or not $\overline{E}$ is itself Skolemised. If $\overline{E}$ is *not* Skolemised then we can only conclude that $\neg p$ holds of *some* constant - with nothing to suggest that this constant should be a. If $\overline{E}$ *is* Skolemised, then we must introduce a *fresh* Skolem constant and can conclude only that $\neg p(b)$, which again says noting about a.

It is of no consequence that in this example $H \models Bot(B, E)$ *happens* to be true, for below is presented a more realistic example due to Yamamoto [Yam96] §4 p6, in which it is *not* true. In the following example it is easily verified that $B \wedge H \models E$, but $Bot(B, E) = odd(0''') \leftarrow even(0)$ and contrary to (12) $H \not\models Bot(B, E)$

$$\begin{aligned} B &= \left\{ \begin{array}{l} even(0) \\ even(X') \leftarrow odd(X) \end{array} \right. \\ E &= \left\{ \; odd(0''') \right. \\ H &= \left\{ \; odd(X') \leftarrow even(X) \right. \end{aligned}$$

In the **reverse direction** the situation is different. Providing it is clearly understood that $\overline{E}$ is the *Skolemised* negation of E[16] (including where it appears in the definition of $\overline{Bot}(B, E)$[17]), and $\overline{H}$ is the *unSkolemised* negation of H[18], and H contains no Skolem constants (from the Skolemisation of E)[19], then the argument is easily recast into a form that is valid:

Given that

$$H \models Bot(B, E) \tag{13}$$

by the law of contraposition

$$\overline{Bot}(B, E) \models \overline{H} \tag{14}$$

but from the definition of Bot(B,E)

$$B \wedge \overline{E} \models \overline{Bot}(B, E) \tag{15}$$

so that by transitivity

$$B \wedge \overline{E} \models \overline{H} \tag{16}$$

and by another application of contraposition

$$B \wedge H \models E \tag{17}$$

[16]as will be required for computational reasons
[17]otherwise (15) may be falsified
[18]otherwise (14) may be falsified
[19]as is required by definition

3.2 Standard Transformation

Consider the task of adding to a set of definite clauses B a clause h such that the combination entails some given clause e. Since logically speaking e is a universally quantified formula, to prove it we must first prove an instance of it ϵ containing arbitrary variables and then apply $\forall$-introduction. And because this ground instance is an implication, by the deduction theorem this is equivalent to adding the body ϵ^- (which is a conjunction of ground atoms) to B to give B' and then proving head ϵ^+ (which is a single ground atom).

As described in previous sections, Progol itself employs precisely this tactic, as do other ILP algorithms that begin by converting the seed example e to a ground Skolemised form ϵ and removing the resulting body to B. In order to justify this process a little notation is introduced, and a proof is given that solutions of the transformed problem are solutions of the orginal problem, providing a little care is taken over the langauge of h. The utility of this transform in ILP is then established.

Notation (B' and e'). Let e be a definite clause; let θ be a substitution that replaces each distinct variable occurring in X with a fresh (Skolem) constant; and let ϵ represent the ground clause $X\theta$. Let ϵ^+ represent the single ground head atom of ϵ; and let ϵ^- represent the conjunction of ground body atoms of ϵ. Now let $B' = B \cup \{\epsilon^-\}$ and $e' = \epsilon^+$

Theorem 3.1. *$B' \wedge h \models e' \iff B \wedge h \models e$ where h contains no Skolem constants*[†]

Proof.

$$\begin{array}{ccc} & B' \wedge h \models e' & \\ \iff & B \wedge h \wedge \epsilon^- \models \epsilon^+ & \text{definition of } B',\ e' \\ \iff & B \wedge h \models \epsilon^- \rightarrow \epsilon^+ & \text{deduction theorem} \\ \iff & B \wedge h \models \epsilon & \text{definition of } \epsilon^+,\ \epsilon^- \\ \iff & B \wedge h \models e & \forall\text{-introduction}^\dagger/\text{instantiation} \end{array}$$

∎

Remark. The process described above should not be confused with the cannonical process of Skolemisation which replaces *existentially* quantified variables by Skolem functions whose arguments are those variables bound by outer universal quantifiers. In a clause all variables are *universally* quantified, and yet it is these variables that must be grounded with fresh constants.

Remark. The condition that h contains no Skolem constants (introduced during the Skolemisation of e) is a precondition of the $\forall$-introduction. This condition can be weakened to require that h be *logically equivalent* to a clause containing no Skolem constants, and possibly further still. The reverse argument holds whether this precondition is met or not.

Theorem 3.2. *$h \models Bot(B', e') \Longrightarrow B \wedge h \models e$ where h contains no Skolem constants*

Proof. From the previous section $h \models Bot(B', e') \Longrightarrow B' \wedge h \models e'$. And from above $B' \wedge h \models e' \iff B \wedge h \models e$. Hence by transitivity $h \models Bot(B', e') \Longrightarrow B \wedge h \models e$ ∎

It should be clear that the implication shown above is not simply *another* approach to IE, it is the approach *actually* used by Progol and related systems. The above formulation simply makes explicit the initial step of converting the seed example e to ground Skolemised form, and then removing the resulting body to B. The next section motivates an alternative formulation of this semantics, which will be applied in later sections to the task of analysing and generalising the Progol5 algorithm.

3.3 Omega Ground Clause

Consider again the task of adding to B a clause h such that the combination entails e. From the previous section this is equivalent to finding an h that when added to B' entails e'. If B' already entails e' then clearly we are done. Otherwise one way to proceed is by constructing the most specific *ground* hypothesis h' that covers e' relative to B', and then generalising this.

And in constructing such an h' we are surely well advised to place in the head of h' only a ground atom that will enable us to prove the required e' from B', and to place in the body only ground atoms that we can already prove from B'. For if we place some other atom in the head then it will not give the consequences required to complete the proof, and if we place some other atom in the body then we will not be able to use the head at all. Now, atoms of the first sort are those that may be *abduced* from B' with goal e', while atoms of the second sort are those that may be *deduced* from B'.

This analysis is extremely intuitive and has the advantage that it does not refer to negation and clearly exposes the abductive and deductive subtasks in the construction of h'. Once formalised below in the semantics of Omega Ground Entailment ($\Omega_g E$), these ideas will serve as a convenient framework in which to analyse Progol5 and reveal a surprising source of incompleteness. The above intuitions will immediately suggest a strategy for eliminating this incompleteness and achieving still further generalisation over Progol-based algorithms.

But first, in order to firmly ground these appealing notions in reality, the Omega Ground Clause is defined and then shown to be equivalent to the MSH!

Definition (Omega Ground Clause).

$$\Omega_g(B,e) \;:=\; \bigvee\{\alpha \in \mathit{GndAtms} \mid B' \wedge \alpha \models e'\} \leftarrow \bigwedge\{\delta \in \mathit{GndAtms} \mid B' \models \delta\}$$

Theorem 3.3. *$h \models \Omega_g(B,e) \iff h \models Bot(B',e')$ where h contains no Skolem constants*

Proof.

$$\begin{array}{ll}
 & h \models \Omega_g(B,e) \\
\iff & h \models \bigvee\{\alpha \in \text{GndAtms} \mid B' \wedge \alpha \models e'\} \leftarrow \bigwedge\{\delta \in \text{GndAtms} \mid B' \models \delta\} \\
\iff & h \models \bigvee\{\alpha \in \text{GndAtms} \mid B' \wedge \overline{e'} \models \overline{\alpha}\} \leftarrow \bigwedge\{\delta \in \text{GndAtms} \mid B' \wedge \overline{e'} \models \delta\} \\
\iff & h \models \bigvee\{\alpha \mid \alpha \in \text{GndAtms},\ B' \wedge \overline{e'} \models \overline{\alpha}\} \vee \bigvee\{\overline{\delta} \mid \delta \in \text{GndAtms},\ B' \wedge \overline{e'} \models \delta\} \\
\iff & h \models \bigvee\{l \in \text{GndLits} \mid B' \wedge \overline{e'} \models \overline{l}\} \\
\iff & h \models Bot(B',e')
\end{array}$$

∎

Remark. A few comments are in order: The penultimate step can be seen as breaking the set $\{l \in \text{GndLits} \mid B' \wedge \overline{e'} \models \overline{l}\}$ into two subsets: one containing the positive literals α, and the other containing the negative literals $\overline{\delta}$. In the preceding step use is made of the deduction theorem on the left, and the following equivalence on the right: $B' \wedge \overline{e'} \models \delta \iff B' \wedge \models \delta$, where e' and δ are both ground atoms and B' is a set of definite clauses. (One direction uses monotonicity and the other the redundancy of the negative clause $\overline{e'}$ in proving a positive consequence δ from the definite clauses B'). Use is made in the previous step of the equivalence $A \leftarrow B \equiv A \vee \overline{B}$.

It should be clear that fundamentally the StartSet algorithm aims at constructing a finite approximation to the set $\{\alpha \in \text{GndAtms} \mid B' \wedge \alpha \models e'\}$[20], while the BottomSet algorithm aims at constructing a finite approximation to the set $\{\delta \in \text{GndAtms} \mid B' \models \delta\}$[21]. Of course in order to do this efficiently the two process must interact with each other and with the mode-declarations. The learning problem is thus seen to consist of four distinct tasks:

- the **standard transformation** of Skolemising e and removing the body to B
- the **abductive** or backward reasoning task of generating candidate head atoms
- the **deductive** or forward reasoning task of generating candidate body atoms
- the **inductive** task of constructing the best hypothesis clause from these atoms

[20]cannonically: the negations of the negative consequences of B and the negation of e
[21]cannonically: the negations the positive consequences of B and the negation of e

3.4 Relation to Progol5

While $\Omega_g E$ is motivated simply and independently of MSH, the two semantics were shown in theorem 3.3 above to be equivalent. The advantage of $\Omega_g E$ is that it does not introduce counter-intuitive negations into the semantics[22], and it clearly exposes the abductive and deductive subtasks in the construction of the MSH. In particular it provides a clear separation of the semantics for StartSet and BottomSet algorithms, and this property is now used to effect a detailed analysis of Progol5. First recall the precise relationship between $\Omega_g E$ and Progol5:

$$\begin{tikzcd}
\{\alpha \in \mathit{GndAtms} \mid B' \wedge \alpha \models e'\} = D \arrow[d, "\begin{array}{l}\text{ground atoms abducible}\\ \text{from B' and e' are}\\ \text{candidates for head}\\ \text{(StartSet)}\end{array}"'] & A = \{\delta \in \mathit{GndAtms} \mid B' \models \delta\} \arrow[d, "\begin{array}{r}\text{ground atoms deducible}\\ \text{from B' are}\\ \text{candidates for body}\\ \text{(BottomSet)}\end{array}"] \\
\text{HEAD} & \text{BODY} \arrow[l, twoheadrightarrow]
\end{tikzcd}$$

It should be clear that the soundness and resource-bounded flounder-free completeness of BottomSet with respect to the set D is guaranteed by the use of SLD resolution. What is less clear, however, is the relation of StartSet to the set A. The following analysis will show that while StartSet is abductively sound, it suffers from an incompleteness that is easily remedied by existing abductive logic programming (ALP) [KKT92] techniques.

Theorem 3.4 (Soundness of StartSet).

$$\alpha \in \mathit{StartSet}(M, B, e, p) \Longrightarrow B' \wedge \alpha \models e'$$

Proof. For convenience let $\epsilon = e'$ and $\Pi = B' \cup \epsilon^* \cup \{b_j^* \twoheadleftarrow b_0^*, \ldots, b_{j-1}, b_{j+1}, \ldots, b_m \mid b_0 \twoheadleftarrow b_1, \ldots, b_j, \ldots, b_m \in B,\ \text{for } j, m > 0\}$. (The selective call-graph optimisation employed by StartSet in its formation of contrapositives is ommitted without consequence, for none of the contrapositives ommitted by StartSet can participate in any computation). Given that the appropriate language constraints are met α can appear in $\mathit{StartSet}(M, B, e, p)$ only if $\Pi \vdash_{sld} \alpha^*$. But since: i) every rule with a *atom in the head has exactly one *atom at the first position in the body, ii) and no other rules contain *atoms, and iii) there is only one *fact and this is ϵ^*, for the query $?\alpha^*$ to succeed from Π then it must be because either $\alpha = \epsilon$ or else because α^* resolved with the head of a rule $\alpha^* \twoheadleftarrow b_1^*, O_1$ and the call $?b_1^*, O_1$ succeeded. By induction the call stack *must* assume the following form:

$$\begin{array}{ll}
? & \alpha^* \\
? & b_1^*, O_1 \\
? & b_2^*, O_2, O_1 \\
 & \vdots \\
? & b_{k-1}^*, O_{k-1}, \ldots, O_1 \\
? & \epsilon^*, O_k, O_{k-1}, \ldots, O_1 \\
? & O_k, O_{k-1}, \ldots, O_1 \\
? & O_{k-1}, \ldots, O_1 \\
 & \vdots \\
 & \square
\end{array}$$

And for this computation to have succeeded, two things are necessary: i) there must be a call path in Π from α^* to ϵ^*

[22]see footnotes 20 and 21

$$\begin{array}{lcl} \alpha^* & \twoheadleftarrow & b_1^*, O_1 \\ b_1^* & \twoheadleftarrow & b_2^*, O_2 \\ & \vdots & \\ b_{k-1}^* & \twoheadleftarrow & \epsilon^*, O_k \end{array}$$

so that B must contain the corresponding clauses

$$\begin{array}{lcl} b_1 & \twoheadleftarrow & \alpha, O_1 \\ b_2 & \twoheadleftarrow & b_1, O_2 \\ & \vdots & \\ \epsilon & \twoheadleftarrow & b_{k-1}, O_k \end{array}$$

and ii) the calls $O_k, O_{k-1}, \ldots, O_1$ must have succeed from B', since every O_j is *free. Hence

$$B' \models \left\{ \begin{array}{lcl} b_1 & \leftarrow & \alpha \\ b_2 & \leftarrow & b_1 \\ & \vdots & \\ \epsilon & \leftarrow & b_{k-1} \end{array} \right\} \models \alpha \rightarrow \epsilon$$

so that

$$B' \wedge \alpha \models \epsilon$$

and

$$B' \wedge \alpha \models e'$$

as required ∎

Theorem 3.5 (*In*completeness of StartSet).

$$\alpha \in \mathcal{L}_M \textit{ and } B' \wedge \alpha \models e' \not\Rightarrow \alpha \in \textit{StartSet}(M, B, e, p)$$

Proof. Proof is by either of the following counter-examples

Example 3.6.

$$M_1 = \{\ \textit{modeh}(1, z)\ \} \quad B_1 = \left\{ \begin{array}{l} w \leftarrow x, y \\ x \leftarrow z \\ y \leftarrow z \end{array} \right\} \quad e_1 = \{\ w\ \} \quad p_1 = \alpha_2 = \{\ z\ \}$$

Example 3.7.

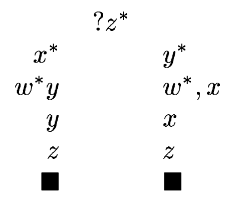
$$M_2 = \{\ \textit{modeh}(1, z)\ \} \quad B_2 = \left\{ \begin{array}{l} x \leftarrow y, z \\ y \leftarrow z \end{array} \right\} \quad e_2 = \{\ x\ \} \quad p_2 = \alpha_1 = \{\ z\ \}$$

in both cases $B' = B$ and $e' = e$ and $B' \wedge \alpha \models e'$ and $\alpha \in \mathcal{L}_M$, but

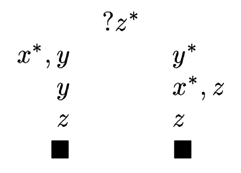
$$\Pi_1 \quad = \quad \left\{ \begin{array}{l} w \leftarrow x, y \\ x^* \leftarrow w^*, y \\ y^* \leftarrow w^*, x \\ x \leftarrow z \\ z^* \leftarrow x^* \\ y \leftarrow z \\ z^* \leftarrow y^* \\ w* \end{array} \right\} \qquad \Pi_2 \quad = \quad \left\{ \begin{array}{l} x \leftarrow y, z \\ y^* \leftarrow x^*, z \\ z^* \leftarrow x^*, y \\ y \leftarrow z \\ z^* \leftarrow y^* \\ x* \end{array} \right\}$$

and $\Pi \not\vdash_{sld} z^*$ so $\alpha \notin \textit{StartSet}(M, B, e, p)$ ∎

Remark. The incompleteness of StartSet is exposed whenever α must be used more than once in a SLD proof of e' from B'. Intuitively StartSet determines if α is an abducible by querying $?\alpha^*$. But if α really *is* abduced then calls to α should succeed, which is where the StratSet algorithm falls short. This situation is shown in the search trees below, both of which fail on the intended abducible z and therefore fail to abduce that atom.



search tree from Π_1



search tree from Π_2

Remark. In the *ground* case completeness may be recovered by adding the *unstarred* form of the query to Π. But this scheme is *less* efficient than the naive approach of adding every possible ground head atom in turn to B' and querying e' to see if it succeeds under SLD resolution.

$$\alpha \in \mathit{StartSet}(M, B, e, p) \iff \Pi \cup \{\alpha\} \vdash_{sld} \alpha^* \text{ and } \alpha \in \mathcal{L}_M$$

Remark. If backtrackable dynamic clause assertion and retraction is available in the underlying SLD engine, then a better solution is obtained by adding the following meta-program to Π in the original StartSet algorithm.

```
:-dynamic bound/0.
:-dynamic delta/1.
q(X) :- \+bound, type_X(X), assert(bound), assert(delta(X)).
q(X) :- bound, delta(X).
```

Here it is assumed that the intended query is q*(X), and for simplicity q has been shown with only one argument. Clearly this program must be generalised in the obvious way by adding the necessary arguments and type calls for q. It must be understood, however, that this solution relies on the cooperation of the type system to ensure that X is grounded before assertion. If this is not the case then the approach is potentially unsound.

Example 3.8 (Fast Food: revisited). That Progol5 does indeed exhibit the incompleteness described above, is demonstrated by the following formalisation of the Fast Food example. When confronted with this input, Progol fails to discover any hypothesis:

```
**********INPUT**********
% CProgol5.0
:- observable(haveMeal/1)?
:- set(h,500)?
:- set(inflate,500)?
% Types
restaurant(mcDonalds).
restaurant(burgerKing).
restaurant(wimpy).
restaurant(theRitz).
% Modes
:- modeh(1,getFries(+restaurant))?
:- modeb(1,specialOffer(+restaurant))?
% Background
haveMeal(R) :- getBurger(R), getFries(R).
getBurger(R) :- getFries(R), specialOffer(R).
getBurger(theRitz).
specialOffer(mcDonalds).
specialOffer(burgerKing).
specialOffer(wimpy).
% examples
haveMeal(mcDonalds).
haveMeal(burgerKing).
haveMeal(wimpy).
:- haveMeal(theRitz).

**********OUTPUT**********
CProgol Version 5.0

[:- observable(haveMeal/1)? - Time taken 0.00s]
[:- set(h,500)? - Time taken 0.00s]
[:- set(inflate,500)? - Time taken 0.00s]
[:- modeh(1,getFries(+restaurant))? - Time taken 0.00s]
[:- modeb(1,specialOffer(+restaurant))? - Time taken 0.00s]
[Testing for contradictions]
[No contradictions found]
[Generalising haveMeal/1]
[Generalising haveMeal(mcDonalds).]
[Generalising haveMeal(burgerKing).]
[Generalising haveMeal(wimpy).]

[Total time taken 0.000s]
```

Remark. This output shows that Progol5 repeatedly fails to find any hypothesis for each example in turn. Incidentally, $haveMeal(R) : -getBurger(R), getFries(R)$ is not treated as an example (as it would in Progol4) because the current implementation of Progol5 only considers *ground* examples.

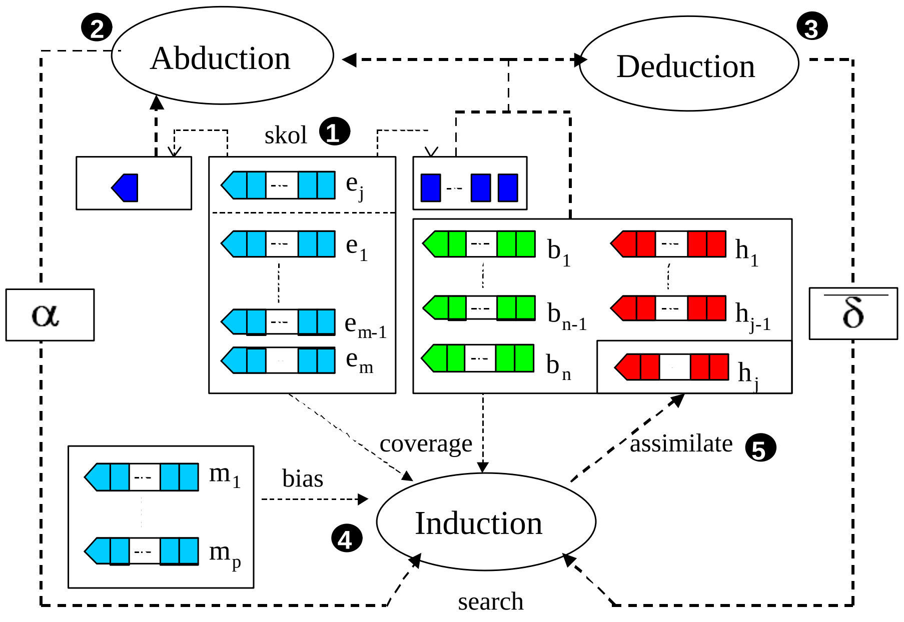
Remark. That the input is not syntactically flawed and that Progol is operating correctly is easily demonstrated, for example, by adding the clause $getBurger(mcDonalds)$ to the background knowledge: in that case the correct hypothesis *will* be found by progol5.

Remark. This incompleteness is related to, but distinct from the incompleteness reported by Yamamoto, or that due to the use of θ-subsumption in the search procedure. Instead it is a less pernicious manifestation that lies *within* the semantics of MSH and can be *avoided* using conventional ALP mechanisms - as will be shown in section 4.

3.5 Relation to Learning Cycle

The diagram below attempts to visualise the interaction of the abductive, deductive and inductive components of the reasoning cycle. Rectangular components denote body atoms, angled components denote head atoms, and a juxtaposition of such components is denotes a clause. The m's denote mode-declarations, the e's examples, the b's background knowledge, and the h's hypotheses. Dotted lines indicate a flow of information between the knowledge base and reasoning mechanisms.

In step 1 the selected seed example e_j is Skolemised and the resulting body atoms are added temporarily to the current background (including any previously induced hypotheses). In step 2 candidate head atoms α are abduced from the augmented background when given the Skolemised head of e_j as the goal. In step 3 candidate body atoms are deduced from the augmented background. In step 4 the optimal hypothesis is found by selecting among the candidate atoms in some systematic way. During this process use may be made of the mode-declarations, and the



coverage of the current hypothesis may be tested. In step 5 the optimal hypothesis is added to the background. A redundancy check should be performed to remove any covered examples or other background clauses, before the next iteration of the cycle is commenced.

4 Nimble

This section further develops the process of generalisation begun in the previous section. Initially a simplified abductive proof procedure is used to simply replace StartSet in the abductive phase of Progol5 to yield Nimble1. Then a fully general ALP mechanism is introduced into the abductive phase and the inductive phase is correspondingly generalised to yield Nimble2. This strategy begins to go beyond the semantics of $\Omega_g E$ and allows the algorithm from a single example to construct multiple hypotheses that would be missed by algorithms such as Progol5 and Nimble1. The additional computation required to find this hypothesis is discussed and a concrete example of such a hypothesis is given. It should be pointed out the contents of this section is of a somewhat preliminary nature.

4.1 Nimble1

Progol is itself a partial realisation of the learning cycle described in the previous section. As a first attempt to create a more general instance of this cycle, Nimble1 simply replaces the StartSet algorithm used by Progol5 with a custom algorithm based on ALP principles. Since only single atoms are required by Progol5 a simplified residue-based ALP procedure is sufficient. One example of such a procedure is the algorithm called Nimble1 below.

As is usual in ALP, the algorithm begins by introducing without loss of generality[23] auxilary predicates in order to ensure that all abducibles are undefined, as this considerably simplifies proceedings. This is accomplished in step 2. We treat as abducible any ground atom that conforms to some head-declaration. The idea is that Δ will contain the abduced atom. Initially Δ is empty and it is assumed that assignments are undone upon backtracking. The usual Prolog-like backtracking and search procedures are assumed along with a **selection rule** R.

[23]see [KKT92] §5.1 p31

Algorithm 4.1 (Nimble1).

```
0. input              given $M$, $B$, $e$
1. initialise         $\Delta = SS = \emptyset$
                      $\mathcal{A}$ is the set of ground atoms subsumed by some scheme in $M_h$
2. abductive          for each predicate $p \in \mathcal{A}$ that is defined in $B'$
   transform              add $p(X, Y, \ldots) \leftarrow \delta_p(X, Y, \ldots)$ to $B'$
                          replace $p$ by $\delta_p$ in $\mathcal{A}$
3. query              initialise call stack with $e'$
4. evaluate           % Begin Goal Reduction Loop
                          select next goal $g$ using selection rule $R$
                          if $g$ is abducible then
                              if $\Delta = \emptyset$ then
 check integrity                  if $B' \cup g \vdash_{sld} false$ **fails finitely** then
 adbuce g                             set $\Delta = g$, succeed on $g$ and continue with next goal
                                  else
                                      fail on $g$ and backtrack
                              else if $\Delta = g$ then
                                  succeed on $g$ and continue with next goal
                              else
                                  fail on $g$ and backtrack
                          else
                              perform 1 step of standard SLD goal reduction
                          if computation fails then backtrack
                          else if computation succeeds then add $\Delta$ to $SS$ and backtrack
                          else continue with next goal
                      % End Goal Reduction Loop
5. output             return $SS$
```

Remark. Nimble1 is complete with respect to ΩE_g in the sense that there exists some selection rule R to compute every possible abducible, however if R is fixed once and for all then the algorithm may flounder on non-ground abducibles or become trapped in infinite computations. These are precisely the same limitations from which Prolog itself is known to suffer.

Remark. Nimble1 solves all of the examples presented so far in this report. In case of examples 3.6 and 3.7, the new search trees are shown below. In both cases the z is correctly abduced. Nimble1 is then be able to use this atom in the head of the MSH. This would for example enable the system to discover the correct hypothesis in the Fast Food example that defeated Progol5.

$?w$	Δ
x, y	$\{\}$
z, y	$\{\}$
y	$\{z\}$
z	$\{z\}$
$\square$	$\{z\}$

new search tree for example 3.6

$?x$	Δ
y, z	$\{\}$
z, z	$\{\}$
z	$\{z\}$
$\square$	$\{z\}$

new search tree for example 3.7

Remark. The algorithm presented above is only intended to give a high-level overview of the computation: as stated above the search and backtracking details have been deliberately left unspecified. This algorithm replaces only the abductive phase of Progol5: the deductive and inductive components remain the same.

4.2 Nimble2

Nimble1 generalised only the abductive phase of Progol5 with a simplified ALP algorithm. Further benefits are possible if one is prepared to use a fully general ALP procedure in the abductive phase while correspondingly generalising the inductive phase - and Nimble2 does precisely this. The key difference is that while Nimble1 only considers single abducibles in its abductive phase, Nimble2 considers sets (conjunctions) of atoms. In this way Nimble2 can learn multiple clauses from a single seed example. Consider the following scenario:

$$B = \left\{ \begin{array}{l} roadTax(P) \twoheadleftarrow hasMOT(P), isInsured(R). \\ gotService(ali).gotService(bob).gotService(chris). \\ gotService(dov).gotService(eve) \\ paidUp(bob).paidUp(chris).paidUp(dov). \\ paidUp(eve).paidUp(frank). \end{array} \right.$$

$$E = \left\{ \begin{array}{l} roadTax(bob).roadTax(chris). \\ roadTax(dov).roadTax(eve). \\ \bot \twoheadleftarrow buyRoadTax(ali). \\ \bot \twoheadleftarrow buyRoadTax(frank). \end{array} \right.$$

$$H = \left\{ \begin{array}{l} hasMOT(P) \twoheadleftarrow gotService(P). \\ isInsured(P) \twoheadleftarrow paidUp(P). \end{array} \right.$$

All of the algorithms discussed so far in this report fail to find any hypothesis. This is because no single ground atom may abduced that would explain any example. Instead it is necessary to abduce two (or in general n) atoms simultaneously and therefore to jointly construct two (or more) MSH clauses with these atoms as the respective heads. The bodies of the MSH will of course be the same, but the bodies of the resulting hypotheses need not. But having strengthened the abductive phase, the inductive phase must be correspondingly strengthened so that these two MSH may be correctly generalised *together*.

Algorithm 4.2 (Nimble2).

```
0. input            given $M$, $B$, $E$
1. cover set        while $E$ not empty
2. seed                 choose seed example $e_i$ from $E$
3. abduction            for each $\Delta_i$ s.t. $\langle B', M_h \rangle \vdash_{asldic} \langle e'_i, \theta, \Delta_i \rangle$
                            for each abduced atom $\delta_{ij} \in \Delta_i$
                                for each $h_k \in M_h$ s.t. $atom(h_k) \preccurlyeq \delta_{ij}$
4. deduction                        construct $BS_{ijk} = BottomSet(M_b, B', \delta_{ij}, h_k)$
5. induction            % find most compressive *set* of clauses $CC$ by generalising the $BS_{ijk}$
                        add $CC$ to $B$
6. cover removal        remove covered examples from $E$
7. output           return
```

Remark. The notation $\langle \Pi, \mathcal{A} \rangle \vdash_{asldic} \langle g, \theta, \Delta \rangle$ means that the goal g succeeds from program Π under abductive SLD resolution with integrity constraints, returning the substitution θ and the set of abducibles Δ. Each abducible is a ground atom whose predicate appears in $\mathcal{A}$. The number of resolutions performed is assumed to be bounded by N_h, and the number of atoms in Δ is assumed to be bounded by N_a.

The increase in expressivity of Nimble2 over existing Progol-based systems is of course paid for by the additional computation required by the multiple clause search algorithm. This is necessary because the two clauses interact with each other. It is not acceptable to generalise individually[24] the components of a *multi-clause-hypothesis* (resulting from a *single* seed example) because the

[24]for example after adding the ground abducibles temporarily to the background knowledge

combination may turn out to be overly general and violate integrity. While the required search is clearly less efficient than a single clause search, the standard pruning mechanisms are still fully applicable.

In a multi-clause search, for each node visited in the subsumption lattice of a given MSH, the optimal node in the lattice of the next must be determined by another search. But fortunately pruning is applicable in *all* such searches. A naive analysis suggests that the costs grow exponentially with the number clauses, but it should be pointed out that compression decreases with the number of clauses considered and this in turn increases the chances of pruning.

A more serious complicating factor is caused by the fact that in general more than one head-declaration may subsume a given abducible, resulting in different input/output patterns of constant propagation and multiple BottomSets. If this is the case every possible permutation must be investigated, which is clearly computationally intensive. It is not yet clear whether this will prove to be a problem in real applications.

As stated above, the maximum number of clauses considered in a multi-hypothesis is bounded by the user-defined system parameter. N_a. If this parameter is set to 1 Nimble1 behaviour will result, while higher values will result in the investigation of multiple-clause hypotheses. A sensible strategy would be to begin searching for 1-clause-hypotheses and then to consider progressively increasing n-clause-hypotheses only as a last resort.

5 Conclusion

It has been known for some time that Most Specific Hypothesis (MSH) generalisation is incomplete with respect to Inverse Entailment (IE), and that this semantic incompleteness is only the most fundamental sense in which algorithms based on MSH generalisation are incomplete. At the computational level additional incompleteness is inevitably introduced, firstly as a result of computing only a finite subset of the MSH, and secondly by considering only the θ-subsumption subset of entailment.

This paper identified an additional source of incompleteness in existing algorithms, and presented an novel algorithm that remedies this incompleteness and further generalises existing algorithms. These benefits were realised by considering the ILP problem in terms of a learning cycle that integrates abductive and inductive reasoning mechanisms. There appears to be considerable potential in the algorithm presented in this paper, although a more thorough analysis is clearly required.

6 Acknowledgements

I would like to thank Alessandra Russo and Krysia Broda for their helpful questions concerning the content of this report and their comments regarding its structure. I would also like to thank Steve Muggleton for our discussions relating to Progol4 and Progol5.

References

[FO00] K. Furukawa and T. Ozaki. On the completion of inverse entailment for mutual recursion and its application to self recursion. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track, 10th International Conference on Inductive Logic Programming*, pages 107–119, 2000.

[KKT92] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[MB00] S.H. Muggleton and C.H. Bryant. Theory completion using inverse entailment. *Lecture Notes in Computer Science*, 1866:130–146, 2000.

[MF01] S.H. Muggleton and J. Firth. Cprogol4.4: A tutorial introduction. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 160–188. Springer-Verlag, 2001.

[MR94] S.H. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[Mug91] S.H. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[Mug95] S.H. Muggleton. Inverse entailment and progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[Mug98] Stephen Muggleton. Completing inverse entailment. In *International Workshop on Inductive Logic Programming*, pages 245–249, 1998.

[Yam96] A. Yamamoto. Improving theories for inductive logic programming systems with ground reduced programs. Technical Report AIDA-96-19, Fachgebiet Intellektik, Fachbereich Informatik, Technische Hochschule Darmstadt, 1996.

[Yam97] A. Yamamoto. Which hypotheses can be found with inverse entailment? In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 296–308. Springer-Verlag, 1997.