Towards constructive induction in first-order predicate calculus

Stephen Muggleton

Wray Buntine

August 1987

Abstract. We present a framework for induction of Horn clauses based around inverting resolution. The framework has its roots in the Duce system for induction of propositional Horn clauses. We motivate the framework by considering shortcomings of previous work and relevant theory.

1 Introduction

2 Review

A number of approaches have been put forward to solve the task of, and tasks related to, Prolog induction. These approaches are unable, however, to introduce new predicates into the language they are supplied. If existing predicates in the language have an intended interpretation, this restricts the class of programs that can be induced from examples.

In this section we review some additional problems with these approaches.

Shapiro's Model Inference System Shapiro's system [8] for inducing Horn clause programs has an extensive theoretical basis. Additional relevant

theory has been developed by Buntine [2]. The programs induced by this system have termination problems because clauses are developed top-down and no demonstration is required of their use in a successful computation. Of course, this can be repaired by modifying the algorithm so that clauses conjectured do have a demonstration of there use in a successful computation. (this is equivalent to making the justifiability assumption [2]).

Mitchell's Version Spaces This approach [4] is a general strategy for induction given a hypothesis space structured using generality. But as a basis for Horn clause induction, the approach is inappropriate. Similar problems would exist for the approach when applied to induction in any general logical framework. Firstly, the approach seems not to adequately account for recursion. Secondly, it is unrealistic to assume that an induction problem can be set up in such a way that a hypothesis can always be represented as a single clause (that is, conjunctive rule). Though, this could be overcome by using some kind of multiple Version Spaces approach. Thirdly, it is usually impossible to narrow the space of hypotheses (Horn clause programs) down to a singleton set, as the Version Spaces approach assumes. This is because, with recursion and function symbols, the hypothesis space is infinite. One result of this is that the hypothesis space will need to be searched heuristically; but the approach offers no guiding principles. Another result is that some representation of the full set of potential hypotheses, under the usual notion of generality, could not normally be maintained. Shapiro avoids this problem by incorporating clause size as well as generalisation (θ-subsumption) in the ordering of hypotheses. Finally, most specific generalisations are not a practical concept to work with in the Horn clause framework [2]. The concept needs, at least, to be weakened by introducing some notion of relevancy, or some restriction on the class of clauses allowed.

3 Basic Principles

In this paper we propose an alternative framework. It is motivated by experience with the Duce [5] system for induction of propositional Horn clauses.

3.1 Inverting resolution

Although it is apparent to many researchers in Machine Learning that there is a strong relationship between deductive theorem-proving mechanisms and inductive inference, this idea has rarely been investigated to any greater depth than to note that the idea of *logical subsumption* or *logical implication* are central to both. One exception to this is Plotkin [6], who on the suggestion of Popplestone, investigated the idea that

> *just as unification was fundamental to deduction, so might a converse be of use in induction.*

From this idea Plotkin went on to develop the concept of *least general generalisation*, or *anti-unification* of literals and clauses.

Unification is a basic idea within Robinson's [7] theory of resolution. Another important concept within this theory is that of the resolution tautology, or rule of inference. Given two clauses we can infer their resolvent which is a specialisation of both clauses. As Plotkin [6] notes

> *It is interesting that ... the similarity between induction and deduction breaks down ... [with anti-unification]. What is useful is not a concept of unification of two clauses, but the deduction principle called resolution.*

We now show that the analogy between deduction and induction can be extended fruitfully, and that in fact the operators used by Duce are merely the

inverse of resolution. Together with a more general method for undoing unification than anti-unification we can use this approach to extend the use of Duce operators to constructive induction within a first-order framework.

3.2 Compaction

The use of compaction as a principle for learning is equivalent to an *a priori* preference for simplicity, i.e. Occam's razor. How can this be justified?

3.3 The Oracle

The Duce algorithm uses an oracle to check the correctness of generalisations and ensure comprehensibility of new terms introduced. Is it possible to give a characterisation of how much information will be required from the oracle with a given number of examples of a logic program? etc.

4 Preliminaries

We assume basic familiarity with resolution and unification. The terminology used is that of Lassez, Maher and Marriott [3].

Terms, atoms (predicate symbols applied to terms), literals (positive or negative atoms), clauses (sets of literals), and Horn clauses (clauses containing less than 2 positive literals) have there usual meaning. The set of varables occurring in any syntactic object o is denoted by $vars(o)$.

A *substitution*, $\theta = \{v_1 \leftarrow t_1, v_2 \leftarrow t_2, \ldots v_n \leftarrow t_n\}$ maps terms to variables. It is applied to a term by simultaneously replacing all occurrences of each v_i in the term by t_i. The set of variables $\{v_1, v_2, \ldots, v_n\}$ is denoted by $domain(\theta)$. The set of variables occurring anywhere in a substitution is denoted $vars(\theta)$. Finally, each variable v_i does not occur in any term t_j. That is, substitutions are idempotent.

A *most general unifier* or *mgu* for two terms or literals, t_1 and t_2, is a most general substitution, θ, for the two terms such that $t_1\theta$ and $t_2\theta$ are identical terms. Two terms or literals are *unifiable* if they have such an mgu.

An *inverse substitution* for a syntactic object o is a set

$$\{(t_1, \{p_{1,1}, \ldots, p_{1,m_1}\}) \rightarrow v_1, \ldots, (t_n, \{p_{n,1}, \ldots, p_{n,m_n}\}) \rightarrow v_n\}$$

in which all t_i are terms in o, all v_i are variables such that $v_i \notin vars(o)$, and $p_{i,j}$ are distinct *places* within the object o. To each inverse substitution, there corresponds a natural substitution that is its inverse, $\{v_1 \leftarrow t_1, v_2 \leftarrow t_2, \ldots v_n \leftarrow t_n\}$. Given the substitution θ, we denote by θ^{-1} an inverse substitution whose natural inverse is θ.

Let C_1 and C_2 be two clauses that have no variables in common. We write the resolvent, or *resolved product*, of C_1 and C_2 as

$$C = C_1 \cdot C_2$$

where $C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$, given that $L_1 \in C_1$ and $L_2 \in C_2$, L_1 and $\overline{L_2}$ are unifiable, and θ is a most general unifier for L_1 and $\overline{L_2}$. L_1 and L_2 are called the literals *resolved on*. Note that the substitution θ can be uniquely factored into component parts θ_1 and θ_2 such that $\theta = \theta_1\theta_2$, $domain(\theta_1) \subseteq vars(C_1)$ and $domain(\theta_2) \subseteq vars(C_2)$. This gives $C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2$.

The resolve product is not unique when more than one pair of literals can be found to resolve on. If only one pair can be found, the resolved product is unique up to a uniform change of variable names.

5 Inversion of a Single Resolution Step

In this section we show how to invert a single resolution step. We do this using first-order equivalents of the absorption and identification operators used in Duce.

First we define the absorption operator and its inverse. In the following two definitions $C = C_1 \cdot C_2$, where the literal resolved on in C_1 is positive.

Definition 1 *Given some clauses C, C_1 and C_2 as described above, and some Horn clause program $P \supseteq \{C, C_1\}$, the* absorption *operator, $Abs(C, C_1)$, transforms P to $P' = P - \{C\} \cup \{C_2\}$.*

Definition 2 *Given some clauses C, C_1 and C_2 as described above, and some Horn clause program $P \supseteq \{C_1, C_2\}$, the* inverse absorption *operator, $Abs^{-1}(C_1, C_2)$, transforms P to $P' = P - \{C_2\} \cup \{C\}$.*

These operators are usually non-deterministic. They are inverse in the following sense: if some application of Abs transforms P to P', then there exists an application of Abs^{-1} transforming P' to P, and vice versa for Abs^{-1} and Abs.

The identification operator, $Ident$, and its inverse are defined analogously, except that the literal resolved on in C_1 is negative.

In order to apply either the absorption or the identification operators, we need to be able to find an inverse to the resolved product. The notion of a resolved quotient is therefore introduced.

We write the *resolved quotient* of C_1 and C_2 as

$$C = C_1/C_2 \tag{1}$$

This must satisfy $C_1 = C \cdot C_2$. In the case of Horn clauses we call $C = C_1/C_2$ an *identificant* of C_1 and C_2 just when the literal resolved on in C_2 is a negative literal. Similarly, for Horn clauses we call $C = C_1/C_2$ an *absorbant* of C_1 and C_2 just when the literal resolved on in C_2 is a positive literal.

Suppose $C_1 = C/C_2$, then though neither C_1 nor C_2 θ-subsume C, they are both more general than C relative to each other (as $C_1 \wedge C_2 \rightarrow C$). In the propositional case as in Duce the resolved quotient of two clauses is unique. For first-order clauses, the resolved quotient is not unique in general.

Let us look in more detail at the case in which we attempt to find a resolved quotient of C and C_1. For some $L_1 \in C_1$, let ϕ_1 be a most general substitution

whose domain is a subset of domain $vars(L_1)$ such that

$$(C_1 - \{L_1\})\phi_1 \subseteq C \tag{2}$$

Let ϕ_2 be a substitution whose domain is a subset of $vars(L_1) - vars(C_1 - \{L_1\})$, L_2 be a literal, and θ_2 be a substitution whose domain is a subset of $vars(L_2)$, such that $\phi_1\phi_2\theta_2$ is an mgu for $\overline{L_1}$ and L_2. $\phi_1\phi_2$ corresponds to θ_1 in the definition of resolved product. Finally, let ${\theta_2}^{-1}$ be an inverse substitution for θ_2 such that $(\overline{L_1}\theta_1){\theta_2}^{-1} = L_2$. In order to invert a single resolution step C_2 is then given by

$$C_2 = (C - (C_1 - \{L_1\})\theta_1 \cup \{\overline{L_1\theta_1}\}){\theta_2}^{-1} \tag{3}$$

Since the choices for L_2 and θ_2 are usually non-unique, as in Duce we would apply a search to find solutions for C_2, and offer these to the oracle with a preference for low complexity clauses. For the special case of finding the absorbant, the problem is simplified somewhat as the choice of L_1 is unique. When $\phi_2 \neq \{\}$, the substitution θ_1 can take on an infinite number of different values. The problem can be avoided by assuming that all substitutions are relevant to the rest of the clause, and do not introduce unnecessary extra function symbols.

Example 1

First we run the resolution step forward. Let

$$\begin{array}{lll} C_1 = & member(H,[H|T]) & \theta_1 = \{H \leftarrow 1, T \leftarrow L\} = \phi_2 and \phi_1 = \{\} \\ C_2 = & a(K) \leftarrow member(1,K) & \theta_2 = \{K \leftarrow [1|L]\} \\ C = & a([1|L]) & \end{array}$$

Given only C and C_1 we first construct $L_1 = member(H,[H|T])$ and so $\phi_1 = \{\}$. Applying equation (3) we get

$$C_2 = (a([1|L]) \leftarrow (member(H,[H|T])\phi_2)){\theta_2}^{-1} \tag{4}$$

where $\phi_2\theta_2$ is an mgu for L_1 and $(L_1\phi_2){\theta_2}^{-1}$.

Note that one trivial solution for equation (4) is $\phi_2 = \theta_2 = \{\}$. The correct solution can only be discovered by testing solutions against the oracle in order of ascending complexity. In the correct solution, the term $[1|L]$ is equated with $[H|T]$ giving $\theta_1 = \{H \leftarrow 1, T \leftarrow L\}$ and ${\theta_2}^{-1} = \{([1|L], \{1, 2\}) \rightarrow X\}$. Applying (4) this yields $C_2 = a(X) \leftarrow member(1, X)$, which is an alphabetic variant of the original C_2.

6 Inversion of multiple resolution steps

In the previous section we showed that there is a way to reverse a single resolution step. Such an inversion inevitably leads to the generalisation of one of the clauses involved. In this section we show that predicate symbols can be reconstructed non-deterministically from the resolvents of multiple resolution steps, as with the use of the Duce operators of *inter-construction* and *intra-construction.*

We first define the (first-order) intra-construction operator and its inverse. They are inverse in the same sense as the absorption operator. Let L be a negative literal in clause A, $CC = \{C_1, \ldots, C_n\}$ and $BB = \{B_1, \ldots, B_n\}$ be two sets of clauses such that $n > 1$ and $B_i = A \cdot C_i$ for $i = 1 \ldots n$.

Definition 3 *Given a clause A and sets of clauses BB and CC as described above, and some Horn clause program $P \supseteq BB$, such that the predicate symbol used in L does not occur in P, the* intra-construction *operator, $Intra(BB)$, transforms P to $P' = P - BB \cup (\{A\} \cup CC)$.*

Definition 4 *Given a clause A and sets of clauses BB and CC as described above, and some Horn clause program $P \supseteq (\{A\} \cup CC)$, such that the predicate symbol used in L occurs only once in A and once in each C_i and nowhere else in P, the* inverse intra-construction *operator, $Intra^{-1}(A, CC)$, transforms P to $P' = P - (CC \cup \{A\}) \cup BB$.*

The inter-construction operator and its inverse are defined analogously, except that the literal L is positive.

Application of the operators can be computed as follows.

Let some clause A containing a literal L be resolved against some set of clauses $CC = \{C_1, \ldots, C_n\}$ producing a set of resolvents $BB = \{B_1, \ldots, B_n\}$ where $n > 1$ such that

$$B_i = (A - \{L\})\theta_{1,i} \cup (C_i - \{L_i\})\theta_{2,i} \tag{5}$$

where $\theta_i = \theta_{1,i}\theta_{2,i} = mgu(L, \overline{L_i})$. If we are given only BB and required to guess A and CC it will be necessary to reconstruct the literal L, that is, to introduce a new predicate symbol into the clauses.

To illustrate this, we now consider in more detail the special case in which each C_i is a unit clause, so $C_i = \{L_i\}$. Firstly we note, from equation (5) and the definition of resolved product,

$$\bigcup_{i=1\ldots n} (vars(\theta_{1,i}) \cap vars(A - \{L\})) \subseteq vars(L) \tag{6}$$

This shows that certain variables must appear in L. Also, for all i, $(A-\{L\})\theta_{1,i} = B_i$. Therefore there is some substitution θ_A and most specific clause B such that

$$(A - \{L\})\theta_A = B \tag{7}$$

and are substitutions ϕ_i such that $\theta_{1,i} = \theta_A\phi_i$ and $B_i = B\phi_i$.

To reconstruct A and CC given only BB in this case, we first construct B and the ϕ_i and choose a ${\theta_A}^{-1}$. So $A - \{L\}$ and $theta_{1,i}$ have now been reconstructed.

Then we choose an L, bearing in mind relation (6). We might assume that L contains as few extra terms as possible. In principle this allows us to construct L non-deterministically. Again, as with the inversion of single-step resolution, we must make some assumptions about the relevance of terms within L in order to avoid an infinite number of solutions.

Finally, we choose L_i under the constraint that $\theta_{1,i}\theta_{2,i}$ is an mgu for L and $\overline{L_i}$. This then yields CC, after choosing ${\theta_{2,i}}^{-1}$ for C_i.

Example 2

As with the previous example, we first run the resolution steps forward. Let

$$A = reverse([A|B], C) \leftarrow reverse(B, D), append(D, [A], C)$$
$$C_1 = append([], E, E)$$
$$C_2 = append([F], G, [F|G])$$

Thus

$$\theta_{1,1} = \{D \leftarrow [], C \leftarrow [A]\} \qquad \theta_{2,1} = \{E \leftarrow [A]\}$$
$$\theta_{1,2} = \{D \leftarrow [F], C \leftarrow [F, A]\} \qquad \theta_{2,2} = \{G \leftarrow [A]\}$$

$$B_1 = reverse([A|B], [A]) \leftarrow reverse(B, [])$$
$$B_2 = reverse([A|B], [F, A]) \leftarrow reverse(B, [F])$$

Given B_1 and B_2 as a starting point, equation (7) gives us

$$(A - \{L\})\theta_A = B = reverse([A|B], [M|N]) \leftarrow reverse(B, O)$$
$$\phi_1 = \{M \leftarrow A, N \leftarrow [], O \leftarrow []\}$$
$$\phi_2 = \{M \leftarrow F, N \leftarrow [A], O \leftarrow [F]\}$$

Making ${\theta_A}^{-1} = \{\}$ means that the variables M, N, A, and O must all appear in the literal L. This can be simplified with the inverse substitution

$${\theta_A}^{-1} = \{([M|N], \{1\}) \rightarrow P\}$$

So by relation (6), L must now contain only the variables A, O and P.
The simplest choice is then $\overline{L} = p(A, O, P)$, where p is

some new predicate symbol. This gives

$$A \;=\; reverse([A|B], P) \leftarrow reverse(B, O), p(A, O, P)$$

Lastly if we assume $\theta_{2,1} = \theta_{2,2} = \{\}$*, then*

$$\begin{aligned} C_1 &= p(A, O, P)\theta_A\phi_1{\theta_{2,1}}^{-1} &&= p(A, [], [A]) \\ C_2 &= p(A, O, P)\theta_A\phi_2{\theta_{2,2}}^{-1} &&= p(A, [F], [F, A]) \end{aligned}$$

In fact the predicate p *is a simplified version of append which appends its first argument onto its second argument to produce its third argument. This is perfectly sufficient for* $reverse$ *to work.*

7 Sufficiency of Duce operators

In this section we show that it is possible to learn any "reasonable" logic program from ground unit clause examples and an oracle, using the operators presented. By "possible to learn", we mean that the program can be found by applying some sequence of operators to the examples. This does not consider the process of search, however.

Before considering the analysis, we first need two new operators. The first removes simple cases of redundancy in the rule set.

Definition 5 *Given clauses* A *and* B *such that* A *is equivalent to* B *except for a change of variable names, and some Horn clause program* $P \supseteq \{A, B\}$*, the* subsumption *operator,* $Subs(A, B)$*, transforms* P *to* $P' = P - \{B\}$*.*

Clearly, $Subs^{-1}$ simply creates a duplicate of a clause in a logic program. The second operator generalises the head of a rule.

Definition 6 *Given a clause A, an inverse substitution θ^{-1} for A effecting terms in the head of A, and some Horn clause program P containing A, the* generalise-head *operator, $GenHead(A)$, transforms P to $P' = P - \{A\} \cup \{A\theta^{-1}\}$.*

Given a logic program P, we denote by $(Abs/Intra/Inter/Ident/Subs/GenHead)*$ (P) the set of logic programs that can be produced by some finite sequence of applications of the Abs, $Intra$, $Inter$, $Ident$, $Subs$ and $GenHead$ operators to P. Likewise, $(Abs/GenHead)*$ for some finite sequence of the Abs and $GenHead$ operators.

We secondly, need to introduce some properties of logic programs that will be used. A deadend clause is when that can never be used in a successful derivation. Of cause this is relative to the context in which the derivation is being constructed.

Definition 7 *We say a logic program P has a* deadend clause *if there exists some clause contained in P that is never used in any successful derivation constructed with clauses only from P.*

Now we define the algorithm $Cigol(Abs, Intra)$ (Cigol is logiC backwards). In the following an *inverse derivation* $E \rightarrow P_1 \rightarrow \ldots \rightarrow P_n$ is a mixed sequence of absorption and intra-construction transformations of the example set E into the logic program P_n. We call P_n an inverse derivative or *id* of E.

Definition 8 *The algorithm $Cigol(Abs, Intra)$ applies an inverse substitution to members of a set of ground unit clauses E to produce E' and returns a set of possible Horn clause programs $H = \{P : P \text{ id of } E'\}$.*

We can see H as being the hypothesis space of an algorithm which returns a single hypothesis. Angluin [1] introduced the notion of a *characteristic sample* set of examples for some language L as being a set of examples which are sufficient to allow the inference of L. Here we use the term somewhat loosely to define the notion of a set of examples which induces a hypothesis space containing a particular logic program.

Definition 9 *Given a logic program P we say that E is a characteristic sample of P for algorithm $Cigol_{(Abs,Intra)}$ if and only if E consists of ground unit clauses and $P \in Cigol_{(Abs,Intra)}(E)$.*

Before showing how to construct a finite characteristic sample for any logic program we will introduce the auxilliary notion of an isolated reference.

Remark 1 *If $Intra^{-1}$ is applied to $P \supseteq (\{A\} \cup CC)$ to produce $P' = P - (CC \cup \{A\}) \cup BB$, where A and $C_i \in CC$ are resolved on an isolated reference L' of p in P and CC is the set of all clauses containing the predicate symbol p in their head then the program P' does not contain the predicate symbol p.*

The following algorithm $Char_{(Abs,Intra)}$ can be used to generates a characteristic sample of a given logic program P.

algorithm $Char_{(Abs,Intra)}$ (P)

let $i = 0$, $P_0 = P$

until P_i contains only unit clauses **do**

if $\exists A$ such that A is an isolated reference to p in P_i

P_{i+1} is the result of applying $Intra^{-1}$ to remove p in P_i

else

P_{i+1} is the result of applying Abs^{-1} to remove reference A in P_i

let $i = i + 1$

done

let $f = i$

E is P_f with all variables skolemised

return(E)

end $Char$

Now we must show that this algorithm will generate a characteristic sample for a class of logic programs.

Theorem 1 *$Char_{(Abs,Intra)}(P)$ is a characteristic sample of some logic program P if and only if every clause within P can be used in some derivation of a unit clause from P.*

Proof. Let $E = Char_{(Abs,Intra)}(P)$. According to definition 6 E is a a characteristic sample of P for $Cigol_{(Abs,Intra)}$ if and only if $P \in Cigol_{(Abs,Intra)}(E)$. Let us assume that E is not a characteristic sample of P.

We will first look at the case in which the **until** loop in $Char_{(Abs,Intra)}$ terminates. According to the loop termination condition, P_f must be a set of unit clauses. Thus after skolemisation, E will be a set of ground unit clauses. Since each step i in the derivation $P \rightarrow \ldots \rightarrow P_f \rightarrow E$ was carried out by either Abs^{-1} or $Intra^{-1}$ it follows that the sequence of transformations $E \rightarrow \ldots \rightarrow P$ is an **inverse derivation** of P from E. Thus since P *id of* E, it follows from definition 5 that $P \in Cigol_{(Abs,Intra)}(E)$, and therefore E is a characteristic sample of P. Thus let us assume that the **until** loop does not terminate.

Let p be some predicate symbol used in P. There will either be no atomic formulae A which reference p in P, or several which do. If there are several, then they will be removed one by one by the **else** statement, with the last reference being removed by the **if** statement, together with all remaining occurrences of p in P (Remark 2). Furthermore referenced predicate symbols will be removed one by one until only unreferenced predicate symbols remain. The **else** statement will now remove all remaining self-references (Remark 1), leaving unit clauses, and the **until** loop will terminate. This contradicts the assumption and completes the proof. □

We now investigate the size of the characteristic sample set for a given logic program.

Theorem 2 *Let $E = Char_{(Abs,Intra)}(P)$ and Ps be the set of referenced predicate symbols in P. The size of the characteristic sample set $|E| = |P| - |Ps|$.*

Proof. From definition 2 Abs^{-1} applies the transformation $P' = P - C_2 \cup C$, and therefore $|P'| = |P|$. From definition 4, $Intra^{-1}$ applies the transformation $P' = (CC \cup \{A\}) \cup BB$, where $|CC| = |BB|$. It follows that for $Intra^{-1}$, $|P'| = |P| - 1$. In the proof of lemma 1 we have shown that referenced predicate symbols are removed one by one using $Intra^{-1}$. All other transformations employ Abs^{-1}. Since there must therefore be $|Ps|$ applications of $Intra^{-1}$ it follows that $|E| = |P| - |Ps|$. □

Not only does this show that characteristic samples are finite, but surprisingly, less examples are needed to induce a logic program than there are clauses in that program. This is counter-intuitive to the normal belief in inductive knowledge engineering, in which we expext to use a large number of examples to induce a small number of rules.

Example 3

We will demonstrate how $Char_{Abs,Intra}$ works on a simple example. In the following let

$$\begin{aligned} P = P_0 = \quad & \{(memss(A,B) \leftarrow member(C,D), member(D,B)), \\ & (member(E,[E|F]) \leftarrow), \\ & (member(G,[H|I]) \leftarrow member(G,I))\} \end{aligned}$$

Thus $Ps = \{member\}$. Applying Abs^{-1} to the second and the third clause we get

$$\begin{aligned} P_1 = \quad & \{(memss(A,B) \leftarrow member(C,D), member(D,B)), \\ & (member(E,[E|F]) \leftarrow), \\ & (member(J,[H,J|K]) \leftarrow)\} \end{aligned}$$

Applying Abs^{-1} to the first and the second clause we get

$$P_2 = \quad \{(memss(C,[L|M]) \leftarrow member(C,L)),$$

$$(member(E,[E|F]) \leftarrow),$$
$$(member(J,[H,J|K]) \leftarrow)\}$$

Applying $Intra^{-1}$ to the first and the second clause we get

$$P_3 = \{(memss(N,[[N|O]|M]) \leftarrow),$$
$$\{(memss(P,[[Q,P|R]|M]) \leftarrow)\}$$

Skolemising we get

$$E = \{(memss(n,[[n|o]|m]) \leftarrow),$$
$$(memss(p,[[q,p|r]|m]) \leftarrow)\}$$

As predicted by theorem 2, $|E| = |P| - |Ps| = 2$.

8 Discussion

In summary, in section 2 we showed that the Duce operators simply represent the inversion of steps in a resolution proof. In the case of general (non-Horn) clauses, only two different operators exist, single-step inversion, and multiple-step inversion. The former lead to generalisations, and the latter to the introduction of new terms. Since the use of Horn clauses introduce an assymetry, each of these operators corresponds to two Horn clause operators, giving the original four Duce operators.

Stephen Muggleton has so far coded a special case of the *absorption* operator and tested it on some simple examples of recursive clause. Meanwhile Wray Buntine has coded the characteristic sample generation algorithm $Char(Abs, Intra)$. One interesting result to come out of the latter encoding is the fact that if we require the characteristic sample generation algorithm to produce larger P_i at

each step (allowing gradual compression in reverse), some logic programs require many more examples than indicated in theorem 2. For instance, naive reverse requires far fewer examples than fast reverse.

Time complexity of logic programs learned by Duce. In the case of learning propositional Horn clauses the time complexity of the learned program was of little consequence, since it is bounded above by the space complexity of the program. This is merely a property of non-looping programs. However no first order logic program containing recursion will have its time complexity constrained in the same way. Is there any way to make a first-order version of Duce search for low time complexity algorithms?

References

[1] D. Angluin. Inference of reversible languages. *JACM*, 29:741–765, 1982.

[2] W. Buntine. Induction of horn clauses: methods and the plausible generalization algorithm. *International Journal of Man-Machine Studies*, 26(4):499–519, 1987.

[3] K. Marriott J.L Lassez, M.J. Maher. Unification revisited. In *Who Knows*. Springer-Verlag, 1987.

[4] T.M. Mitchell. Generalisation as search. *Artificial Intelligence*, 18:203–226, 1982.

[5] S.H. Muggleton. Duce, an oracle based approach to constructive induction. In *IJCAI-87*, pages 287–292. Kaufmann, 1987.

[6] G.D. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, August 1971.

[7] J.A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12(1):23–41, January 1965.

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