Machine Invention of First-Order Predicates by Inverting Resolution

STEPHEN MUGGLETON (STEVE@TURING.AC.UK)
The Turing Institute, George House,
36 Hanover Street, Glasgow G1 2AD, UK.

WRAY BUNTINE (WRAY@SESG.OZ)
The New South Wales Institute of Technology,
P.O. Box 123, Broadway, NSW2007, Australia.

Abstract

It has often been noted that the performance of existing learning systems is strongly biased by the vocabulary provided in the problem description language. An ideal system should be capable of overcoming this restriction by defining its own vocabulary. Such a system would be less reliant on the teacher's ingenuity in supplying an appropriate problem representation. For this purpose we present a mechanism for automatically inventing and generalising first-order Horn clause predicates. The method is based on inverting the mechanism of resolution. The approach has its roots in the Duce system for induction of propositional Horn clauses. We have implemented the new mechanism in a system called CIGOL. CIGOL uses incremental induction to augment incomplete clausal theories. A single, uniform knowledge representation allows existing clauses to be used as background knowledge in the construction of new predicates. Given examples of a high-level predicate, CIGOL generates related sub-concepts which it then asks its human teacher to name. Generalisations of predicates are tested by asking questions of the human teacher. CIGOL generates new concepts and generalisations with a preference for simplicity. We illustrate the operation of CIGOL by way of various sessions in which auxiliary predicates are automatically introduced and generalised.

1 Introduction

Most learning systems, whether inductive or explanation-based, produce generalisations from examples. However, as Utgoff (Utgoff, 1986), Rendell (Rendell, 1985) and others have pointed out, the performance of such systems is strongly biased by the vocabulary used within the foreground examples and any supplied background domain knowledge. Although conceptual clustering algorithms such as those of Michalski and Stepp (Michalski and Stepp, 1982) and Rendell (Rendell, 1985) have broken ground within this area, these methods have tended to be limited to the construction of new taxonomic clauses within non-relational formalisms.

Discovery systems, such as Lenat's AM program (Lenat, 1979), represent a second approach to the construction of new predicates. AM independently proposed new concepts based on machine investigation within a mathematical domain. The discovery process was guided by a set of heuristics of "interestingness", and not by any external goal. Other work on discovery (Langley et al., 1983; Epstein, 1987) shares this property of being only weakly directed by general heuristics.

It is in this aspect of directedness or goal orientation that the work described in this paper differs substantially from that of work on machine discovery. For this reason we prefer to call the new approach *machine invention.* Thus rather than employing a set of *ad hoc* heuristics, the learning program continually assesses the utility of new invented concepts in terms of more absolute parameters of performance improvement.

The present approach has its roots both in the previously mentioned Duce system for induction of propositional Horn clauses (Muggleton, 1987) and in Sammut's MARVIN (Sammut, 1981). MARVIN uses a single operator to generalise examples within a first-order Horn clause representation. Meanwhile, Duce applies the same operator together with an additional, related set of transformations to introduce and generalise predicates within a propositional logic formalism. On analysis, Duce's transformations proved to be simple inversions of Robinson's (Robinson, 1965) mechanism of resolution (Muggleton, 1991). In Section 2 we illustrate a Prolog implementation of this mechanism called CIGOL (LOGIC backwards). In Sections 3 and 4 we give the theoretical basis of CIGOL and demonstrate that by extending the notion of inverse resolution to first-order logic, it is possible to produce powerful learning mechanisms capable of inventing and generalising first-order relational predicates. In Section 5 we describe implementation details of CIGOL.

Although CIGOL is essentially an inductive learning algorithm, the approach is related to the explanation-based learning (EBL) approach to machine learning. Firstly, like explanation-based systems described by DeJong and Mooney (DeJong and Mooney, 1986) and Mitchell, Keller and Kedar-Cabelli (Mitchell et al., 1986), CIGOL carries out intensive analysis of single examples with reference to an existing background theory. This leads to a very efficient use of example material. In (Muggleton and Buntine, 1988) we have shown that the minimum number of ground unit examples required to learn a theory containing N clauses is merely $2(N - P)$ where P is the number of predicates invented by the system. Secondly, Kedar-Cabelli and McCarty (Kedar-Cabelli and McCarty, 1987) have shown that mechanisms based on resolution can be used to produce a powerful and simple framework for explanation-based learning. This suggests a possible route for integration with the mechanisms described here. Towards these ends, Wirth (Wirth, 1988) has used our method of inverse resolution to deal with part of the incomplete theory problem in EBL. There are two senses in which any logical theory can be incomplete: the theory can have insufficient coverage for existing predicates, or, alternatively, it can have an insufficiently rich vocabulary for describing the domain. In this paper we attack both these problems.

```
| ?- cigol.
!- member(blue,[blue]).
!- member(eye,[eye,nose,throat]).
   TRUNCATION (-20)
     Is member(A,[A|B]) always true? y.
!- member(red,[blue,red]).
   TRUNCATION (-15)
     Is member(A,[B|C]) always true? n.
!- member(2,[1,2,3,4,5,6]).
   TRUNCATION (-33)
     Is member(A,[B,A|C]) always true? y.
   ABSORPTION (-1)
     New clauses:[(member(A,[B|C]):-member(A,C))]
     cover new facts: [member(A,[B,C,A|D]),..]
   Are new clauses always true? y.
!- show_clauses.
   member(A,[A|B]).
   member(A,[B|C]):-member(A,C).
```

Figure 1: CIGOL learns list membership

2 CIGOL Sessions

In this section we illustrate the operation of CIGOL by way of sessions in which it learns and invents various predicates.

2.1 List membership

Figure 1[1] shows a session in which CIGOL learns list-membership. A session involving predicate invention is left to Section 2.2.

User input is shown in bold. On the first line CIGOL is called from the Prolog prompt level. CIGOL prompts the user for an example with "!-". The first example states that "blue" is a member of the list [*blue*]. Note that although this is a ground unit clause, examples can take the form of arbitrary non-ground Horn clauses. Since there are no other examples, CIGOL returns the prompt. Given the second example, CIGOL applies the truncation operator (see Section 4.3) and asks if the least general generalisation member(A,[A|B]) is true, i.e., is something a member of a list if it is the first element? CIGOL's questions of this form should be taken as universally quantified. The number (-20) after the word TRUNCATION represents the compaction produced in terms of clause size (Section 5). The user indicates that the generalisation is true. In the next two examples the user shows CIGOL that something is a member of a list if it is the second element. At first CIGOL over-generalises with the hypothesis member(A,[B|C]), i.e. anything is a member of any list. The user rejects this.

[1]Edinburgh Prolog syntax (Clocksin and Mellish, 1981) is used throughout the examples.

However, the next truncation member(A,[B,A|C]), is correct. CIGOL uses this new clause as the basis for constructing an inverse proof, finding that it can apply the absorption operation (see Section 4.1) to get the general recursive clause (member(A,[B|C]) :- member(A,C)). CIGOL uses a depth-bounded SLD theorem-proved to show the user that the additional, as yet unknown fact, member(A,[B,C,A|D]) can be derived using the new clause. The user confirms this generalisation. CIGOL cannot generalise further and so returns to the prompt. In reply to the user's typing "show-clauses", CIGOL shows the clauses which are known to be true.

2.2 Arch problem

In (Sammut and Banerji, 1986), the operation of MARVIN is illustrated by its ability to learn the description of an arch. Where MARVIN had to be taught each sub-concept involved separately, CIGOL is able to invent these auxiliary subproblems itself. Figure 2 shows the interaction involved in this process.

The specific form of arch to be learned in this example has two columns on either side consisting of a mixture of pairwise matching bricks and blocks. The arch is topped with a beam. Arches are represented symbolically by terms consisting of triples of the form (Column 1, beam, Column 2). In the session of Figure 2, the user consults, without generalisation, a file of clauses called "arch". In general, the consult facility used is for loading libraries of problem-specific background knowledge. Having done this, typing "show-clauses" displays the ground unit clauses which have been consulted. The user then types in a new example of an arch. This leads to three unsuccessful truncations followed by an intra-construction (see Section 4.2). The intra-construction suggests a new predicate, p110, showing the call to p110 together with four instances of the new predicate. The user recognises and names p110 as "column". CIGOL now applies the absorption operator to produce a recursive generalisation of the two clauses for column, which the user confirms. CIGOL now tries to generalise this recursive clause further with a doubly recursive clause. Although this new generalisation is strictly correct, it is over-complicated and covers no new facts. The user thus replies with "i", telling CIGOL to ignore this generalisation.

The absorption operator is now applied again to produce another recursive clause for column, which the user confirms. The intra-construction operator now introduces a new predicate, p190, which allows the recursive clauses for column to be combined into a single clause. The user names the new predicate "brick-or-block", representing a new type of object. CIGOL finds no more operator applications and thus returns to the prompt. Lastly, the user asks the system to display all the clauses. This final set of clauses is a complete, correct and non-redundant concept definition.

In Section 3 and 4 we give the theoretical basis for CIGOL's inverse resolution operators of truncation, absorption and intra-construction.

```
| ?- **cigol.**
!- **[-arch].**
!- **show_clauses.**
     arch([],beam,[]).
     arch([block],beam,[block]).
     arch([brick],beam,[brick]).
!- **arch([block,brick],beam,[block,brick]).**
   TRUNCATION (-71)
     Is arch(A,beam,A) always true? **n.**
   TRUNCATION (-65)
     Is arch([A|B],beam,[A|B]) always true? **n.**
   TRUNCATION (-53)
     Is arch([block|A],beam,[block|A]) always true? **n.**
   INTRA-CONSTRUCTION (-32)
       arch((A,beam,A)):-p110(A).
       p110([]).
       p110([block]).
       p110([block,brick]).
       p110([brick]).
     What shall I call p110? **column.**
   ABSORPTION (-2)
     New clauses:[(column([block|A]):-column(A))]
     cover new facts: [column([block,block]),column([block,block,brick]),..]
   Are new clauses always true? **y.**
   ABSORPTION (-10)
     New clauses:[(column([A|B]):-column([A,brick]),column(B))]
     cover new facts: [...]
   Are new clauses always true? **i.**
   ABSORPTION (-2)
     New clauses:[(column([brick|A]):-column(A))]
     cover new facts: [column([brick,block]),column([brick,brick]),..]
   Are new clauses always true? **y.**
   INTRA-CONSTRUCTION (-3)
       column(([A|B])):-p190(A), column(B).
       p190(block).
       p190(brick).
     What shall I call p110? **brick_or_block.**
!- **show_clauses.**
     arch(A,beam,A) :- column(A).
     brick_or_block(block).
     brick_or_block(brick).
     column([]).
     column([A|B]):-brick_or_block(A),column(B).
```

Figure 2: Arch problem

3 Preliminaries

Although we assume basic familiarity with resolution and unification, in this section we briefly review some of the concepts involved. The terminology developed will be used in Section 4 to describe inverse resolution.

3.1 Unification

We use terminology similar to that of Lassez, Maher and Marriott (Lassez et al., 1986). Terms, atoms (predicate symbols applied to terms), literals (positive or negative atoms), clauses (sets of literals) and Horn clauses (clauses containing one positive literal) have their usual meaning. The set of variables occurring in any syntactic object, o, is denoted by vars(o). A substitution $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$, uniquely maps terms to variables. It is applied to a term by replacing all occurrences of each v_i by the corresponding term t_i. The set of variables $\{v_1, \ldots, v_n\}$ is denoted by domain(θ). A unifier for two terms or literals t_1 and t_2, is a substitution θ, such that $t_1\theta = t_2\theta$. The substitution θ is a *most general unifier* or *mgu* of t_1 and t_2 if and only there is no other unifier θ' for which the unified term $t_1\theta'$ is more general than $t_1\theta$. We say that two terms or literals are *unifiable* if they have an *mgu*.

For the purpose of inverting resolution steps we will introduce the notion of an inverse substitution. Given a term or literal t and a substitution θ, there exists a unique inverse substitution θ^{-1} such that $t\theta\theta^{-1} = t$. Whereas the substitution θ maps terms to variables within t, the inverse substitution θ^{-1} maps variables in t to terms in $t\theta$. Thus if

$$\theta = \{v_1/t_1, \ldots, v_n/t_n\}$$

we denote the corresponding inverse substitution by

$$\theta^{-1} = \{(t_1, \{p_{1,1}, \ldots, p_{1,m_1}\})/v_1, \ldots, (t_n, \{p_{n,1}, \ldots, p_{n,m_n}\})/v_n\}$$

in which $p_{i,m}$ are the *places* (see below) at which the variables v_i are found within t. Inverse substitutions are applied by replacing all t_i at places $p_{i,1}, \ldots, p_{i,m_i}$ within t by v_i. Places within terms or literals are denoted by n-tuples and defined recursively as follows. The term at place $\langle a_1 \rangle$ within $f(t_1, \ldots, t_n)$ is t_{a_1}. The term at place $\langle a_1, \ldots, a_m \rangle$ within $f(t_1, \ldots, t_n)$ is the term at place $\langle a_2, \ldots, a_m \rangle$ in t_{a_1}. The definition of place can easily be extended to cover places within clauses by assuming a fixed ordering on literal within a clause.

Example 1 *If literal L = likes(A,brother(A)) and $\theta = \{A/john\}$ then $L\theta$ = likes(john,brother(john)) and $\theta^{-1} = \{(john, \{\langle 1 \rangle, \langle 2, 1 \rangle\})/A\}$. $\langle 1 \rangle$ and $\langle 2, 1 \rangle$ are the places within L at which variable A is found. Thus $L\theta\theta^{-1}$ = likes(A,brother,A) = L.*

Given two terms or literals t_1 and t_2 which have no variables in common, we say that the substitution θ is their θ-difference when $t_1\theta = t_2$ and domain(θ) $\subseteq$ vars(t_1). The θ-difference is undefined otherwise. When it is define it is written as the infix relation $\theta = t_1 -_\theta t_2$. In fact the θ-difference between two terms is unique and can be defined recursively as follows.

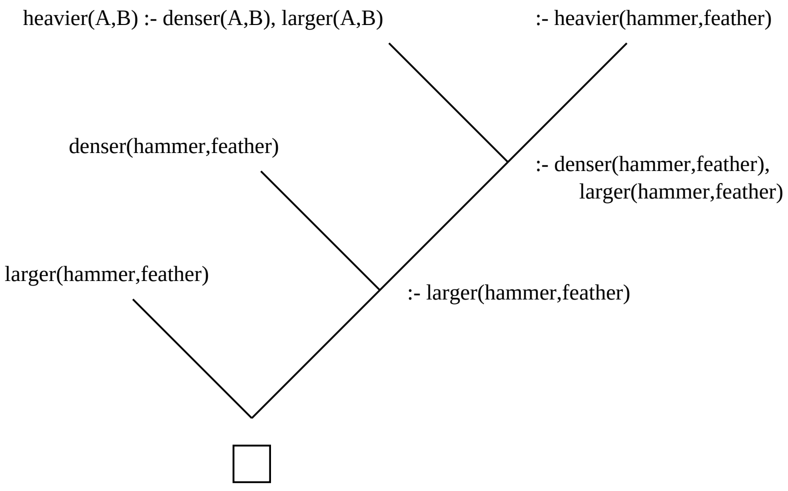
heavier(A,B) :- denser(A,B), larger(A,B)
:- heavier(hammer,feather)
denser(hammer,feather)
:- denser(hammer,feather), larger(hammer,feather)
larger(hammer,feather)
:- larger(hammer,feather)

Figure 3: A refutation tree

$v_1 -_\theta\ t_2 = \{v_1/t_2\}$ if v_1 is a variable.
$f(r_1,..,r_n) -_\theta\ f(s_1,..,s_n) = \bigcup(r_i -_\theta\ s_i)$ for $1 \leq i \leq n$ if f is a predicate or function symbol with arity n.
$t_1 -_\theta\ t_2$ is undefined otherwise.

When $t_1 {-}\theta\ t_2$ is defined we say that t_1 *subsumes* t_2.

Example 2 *The term t_1 = plus(A,B) subsumes the term t_2 = plus(3,4) since $t_1 {-}\theta\ t_2 = \{A/3, B/4\}$.*

3.2 Resolution steps

Resolution allows us to infer a new clause C from two given clauses C_1 and C_2. Let C_1 and C_2 be two clauses with no variables in common. Let L_1 and L_2 be literals within C_1 and C_2 respectively such that θ is the *mgu* of $\overline{L_1}$ and L_2. We say that L_1 and L_2 are the literals *resolved on* and write the resolvent, or *resolved product* of C_1 and C_2 as $C = C_1 \cdot C_2$ where $C = (C_1 - \{L_1\})\theta \cup (C_2 - \{L_2\})\theta$. 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(θ_2) $\subseteq$ vars(C_2). This gives

$$C = (C_1 - \{L_1\})\theta_1 \cup (C_2 - \{L_2\})\theta_2 \qquad (1)$$

3.3 Resolution proofs

Resolution is used for deriving the consequences of a logical theory. Given a theory T, C is a consequence of T (or $T \rightarrow C$) if only and only if $T \wedge \overline{C}$ can be shown to be false. A resolution proof consists of a series of resolution inference steps which generate the empty or false clause given $T \wedge \overline{C}$ as input. Such a proof is often represented graphically as a binary tree.

Example 3 *If T is the theory*

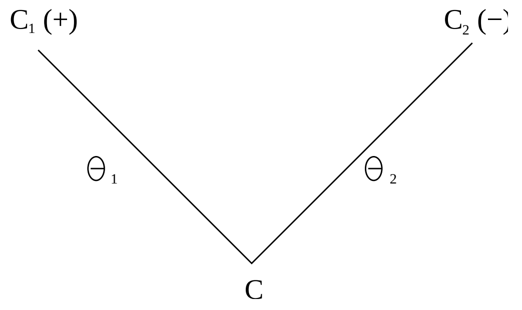

Figure 4: A single resolution step

larger(hammer, feather)
denser(hammer, feather)
heavier(A,B) :- denser(A,B), larger(A,B)

and $\overline{C}$ is the goal

:- heavier(hammer, feather).

then Figure 3 shows a refutation tree for $\overline{C}$. The small open square at the root of the tree represents the empty clause.

4 Inverting resolution

CIGOL employs three types of operators to construct theories from examples. "V" operators generalise existing clauses without introducing new predicates. "W" operators introduce new auxiliary predicates. Lastly the truncation operator is a boundary case "W" operator which generalises unit clauses by applying Plotkin's least general generalisation relation (Plotkin, 1971).

4.1 The "V" operators

The tree in Figure 3 can be seen as being made up of a number of connected "V"s, in which each "V" represents a single resolution inference. Figure 4 shows the general form of these "V"s. Resolution is able to derive the clause the base of the "V" given the clause on the other arm and the clause at the base. In Figure 4, the literal resolved on is positive (+) in C_1 and negative (-) in C_2.

Note that within any theory containing C_1 and C_2, C is redundant since it is directly implied by C_1 and C_2. Owing to the fact that C can be discarded the "V" operators are capable of leading to simplification and compaction of theories.

The *absorption* operator in CIGOL constructs C_2 given C_1 and C. Within the propositional Duce system the *identification* operator constructs C_1 given C_2 and C. The identification operator has not yet been implemented in CIGOL. These two operators together are called the "V" operators.

In order to apply either the absorption or the identification operators we need to be able to find an inverse of the resolved product. The notion of a *resolved quotient* is therefore introduced. With reference to the clauses in Figure 4, we write the *resolved quotient* of C and C_1 as $C_2 = C/C_1$. This must satisfy $C = C_1 \cdot C_2$ (see Section 3.2). In the propositional Horn clause case, as in Duce, the resolved quotient of two clauses is unique under the assumption of *separability* explained below. For first-order clauses, the resolved quotient is not unique in general. However, we can find exact constraints on this quotient by considering equation (1) (Section 3.2). By simple algebraic manipulation of (1), we get

$$C_2 \quad = \quad (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \{L_2\} \tag{2}$$

We have made the assumption here that the clauses $(C_1 - \{L_1\})\theta_1$ and $(C_2 - \{L_2\})\theta_2$ contain no common literals. We make this *separability* assumption throughout the paper. Now, since $\theta_1\theta_2$ is the mgu of $\overline{L_1}$ and L_2, we know that $\overline{L_1}\theta_1 = L_2\theta_2$ and thus

$$L_2 \quad = \quad \overline{L_1}\theta_2\theta_2^{-1} \tag{3}$$

Substitution (3) into (2) we get

$$\begin{aligned} C_2 \quad &= \quad (C - (C_1 - \{L_1\})\theta_1)\theta_2^{-1} \cup \overline{L_1}\theta_1\theta_2^{-1} \\ &= \quad (C - (C_1 - \{L_1\})\theta_1 \cup \overline{L_1}\theta_1)\theta_2^{-1} \end{aligned} \tag{4}$$

Note that the constraints on inverting resolution described by (4) are not restricted to Horn clauses, but hold for arbitrary first-order clauses. Given only C and C_1 there are three sources of indeterminacy within equation (4). Namely, L_1, θ_1 and θ_2^{-2}. In order to reduce this indeterminacy, we first of all assume that C_1 is a unit clause, i.e. $C_1 = \{L_1\}$. This simplifies (4) to $C_2 = (C \cup \{\overline{L_1}\}\theta_1)\theta_2^{-1}$ and reduces the indeterminacy to the choice of θ_1 and θ_2^{-1}. Let us imagine constructing θ_2^{-1} non-deterministically. From the definition of an inverse substitution (Section 3.1), we must decide which terms within $(C \cup \{\overline{L_1}\}\theta_1)$ map to distinct variables. Let term t in C and $t_1\theta_1$ in $\{\overline{L_1}\}\theta_1$ be mapped to the same variable v within θ_2^{-1} where t_1 is a term within $\{\overline{L_1}\}$. This implies that $t_1\theta_1 = t$. Clearly in this case $t_1 -_\theta t$ is defined and is a subset of θ_1. This suggests a non-deterministic algorithm for hypothesising C_2 given C and $C_1 = \{L_1\}$ (algorithm 1). Note that in algorithm 1, by a partition Π of a set S we mean a set of sets such that no two blocks (elements) of Π have any common elements, and the union of the blocks of Π is equal to S.

Algorithm 1 *A non-deterministic algorithm for computing absorption*

```
Input: clauses $C$ and $C_1 = \{L_1\}$
    Let $TP = \{(t,p) | t$ is a term found at place $p$ in $(C \cup \{\overline{L_1}\})\}$
    Let $TP'$ be a subset of $TP$
    Construct a partition $\Pi$ of $TP'$ as follows
```

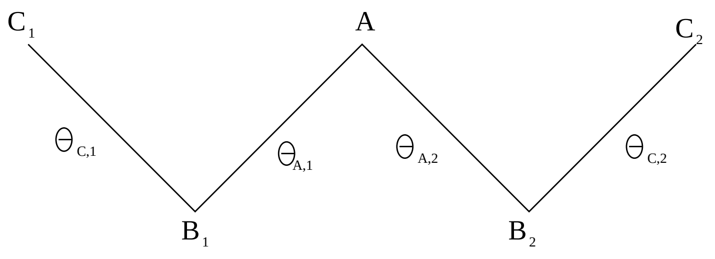

Figure 5: Two resolution steps with common clause A

```
        Let each block $B = \{(r,p_1),..,(r,p_n)\} \cup \{(s,q_1),..,(s,q_m)\}$
            of $\Pi$ be such that $s$ subsumes $r$ and all
            $(r,p_i)$ correspond to terms from $C$ and
            $(s,q_j)$ correspond to terms from $\{\overline{L_1}\}$
        Let $\theta_1 = \bigcup(s -_\theta r)$ for all blocks $B$ as above
        Let $\theta_2^{-1} = \{(r,\{p_1,..,p_n,q_1,..,q_m\})/v | \text{for all blocks } B \text{ as above}\}$ where
            all $v$ are distinct variables not found in $(C \cup \{\overline{L_1}\})$
    Output: $C_2 = (C \cup \{\overline{L_1}\theta_1\})\theta_2^{-1}$
```

Since Algorithm 1 is non-deterministic, it is necessary to reformulate it as a search-based algorithm in order to execute it. In Section 5 we describe CIGOL's best-first search implementation of Algorithm 1. The search uses simplicity or compaction as a heuristic.

Example 4 *Let $C = (A < s(s(A)))$ and $C_1 = (B < s(B))$, where $s(X)$ means the successor of X. Then*

$(C \cup \{\overline{L_1}\}) = ((A < s(s(A))) : -(B < s(B)))$ *and*
$TP = \{(A, \langle 1,1\rangle), (A, \langle 1,2,1,1\rangle), (s(A), \langle 1,2,1\rangle), (s(s(A)), \langle 1,2\rangle),$
$(B, \langle 2,1\rangle), (B, \langle 2,2,1\rangle), (s(B), \langle 2,2\rangle)\}$

Non-deterministically choose $TP' = \{(s(s(A)), \langle 1,2\rangle), (s(B), \langle 2,2\rangle)\}$. *Then*
$\theta_1 = \{B/s(A)\}$,
$\theta_2^{-1} = \{(s(s(A)), \{\langle 1,2\rangle, \langle 2,2\rangle\})/D\}$ *and*
$C_2 = ((A < D) : -(s(A) < D))$

Note that this satisfies the resolution relation $C = C_1 \cdot C_2$.

4.2 The 'W' Operators

By combining together two resolution 'V's back-to-back we get a 'W' of the form shown in Figure 5.

Assume that C_1 and C_2 resolve on common literal L within A to produce B_1 and B_2. The methods described in this section construct the clauses A, C_1 and C_2 given B_1 and B_2. The 'W' operators are called *intra-construction* when L is negative and *inter-construction* when L is positive. Note that since the common literal L in A is resolved away, the clauses A, C_1 and C_2 can contain a literal with a predicate symbol not found in B_1 and B_2. It is in this that new predicates are *invented* by the 'W' operators. Although both

‘W’ operators were implemented within Duce, CIGOL presently only uses the *intra-construction* operator whom we will discuss in this section.

The ‘W’ in Figure 5 can be extended to multiple clauses as follows. Let $BB = \{B_1, \ldots, B_n\}$ and $CC = \{C_1, \ldots, C_n\}$ be two sets of clauses, and L a negative literal in A such that $B_i = A \cdot C_i$ resolved on L. Applying equation (1) we get

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

where $\theta_{A,i}\theta_{C,i}$ is the MGU of $\overline{L}$ and L_i, i.e.,

$$\overline{L}\theta_{A,i} = L_i\theta_{C,i} \tag{6}$$

We make the simplifying assumption that

$$C_i = \{L_i\} \tag{7}$$

Applying this to (5) we get

$$B_i = (A - \{L\})\theta_{A,i} \tag{8}$$

Thus let us choose a clause $B = (A - \{L\})$ such that it is a common generalisation of all clauses B_i. Given such a choice for B we get

$$\theta_{A,i} = B -_\theta B_i \tag{9}$$

From equation (6) we can see that $vars(\overline{L}) \subseteq domain(\theta_{A,i})$ for $1 \leq i \leq n$. The simplest choice for $\overline{L}$ is $\overline{L} = p(v_1, .., vm)$ where $\{\} = \bigcup domain(\theta_{A,i})$ and p is a new predicate symbol. Clearly this is a most general literal, and thus *overlineL* must subsume L_i. Hence unifying $\overline{L}$ and L_i gives L_i. But, according to (6) unifying $\overline{L}$ and L_i gives $L_i\theta_{C,i}$, so $\theta_{C,i}$ must be empty. Applying this fact to (7) and (6) we get

$$C_i = \{L_i\} = \{\overline{L}\}\theta_{A,i} \tag{10}$$

Since we are constructing the definition of a new predicate it is worth checking that we are not introducing *irrelevant* terms into the definition. To explain what is meant here, imagine there is a place $\langle j \rangle$ such that for all L_i the term at place $\langle j \rangle$ shares no variables with the remainder of L_i. Removing the variable v_j from $\overline{L}$ and the corresponding term from C_i will not affect the semantics of any subsequent predicate based on p. Since there is not room here to prove this, we merely demonstrate the effect of this definition of irrelevance in Example 5 below. Lastly, if we rearrange (8) we get

$$\begin{align} A &= B_i\theta_{A,i}^{-1} \cup \{L\} \tag{11} \\ &= B \cup \{L\} \tag{12} \end{align}$$

Equations (9), (10) and (12) give us the basis for the following non-deterministic algorithm.

Algorithm 2 *A non-deterministic algorithm for computing intra-construction*

```
Input: clauses $BB = B_1, .., B_n$
  Let $B$ be a generalisation of the clauses in $BB$ such that
  $\theta_{A,i} = B -_\theta B_i$
  Construct $\overline{L} = p(v_1, .., v_m)$ where $\{v_1, .., v_m\}$
    is the relevant subset of
  $\bigcup domain(\theta_{A,i})$ and $p$ is a new predicate symbol
Output: $A = B \cup \{L\}$ and $C_i = \{\overline{L}\}\theta_{A,i}$
```

Example 5 *Imagine we are given as input the clauses* $BB = \{B_1, B_2\}$ *where*

$$\begin{aligned} B_1 &= (min(D, [s(D)|E])) : -min(D, E) \\ B_2 &= (min(F, [s(s(F))|G])) : -min(F, G). \end{aligned}$$

Here $min(X, Y)$ *should be taken to mean that* X *is the minimum of the list of numbers* Y*. If we let* $B = (min(H, [I|J]) : -min(H, J))$ *then*

$$\begin{aligned} \theta_{A,1} &= \{H/D, I/s(D), J/E\} \\ \theta_{A,2} &= \{H/F, I/s(s(F)), J/G\} \end{aligned}$$

Since none of the terms substituted for J *contain variables present within any other substitutions, it will be irrelevant within the definition of* C_i*. Thus*

$$\begin{aligned} \overline{L} &= p(H, I) \\ A &= (min(H, [I|J]) : -min(H, J), pl(H, I)) \\ C_1 &= (p(D, s(D))) \end{aligned}$$

and

$$C_2 = (p(F, s(s(F))))$$

In fact p is the numeric comparison relation "<", where C_1 *and* C_2 *are two instances of this relation.*

4.3 The truncation operator

The empty clause is found at the root of every refutation tree (see Figure 3). However, the operators so far described do not deal with the case in which the empty clause is found at the base of 'V' or 'W'. The truncation operator described in this section deals with this case. Figure 6 shows such a 'W'. Given the unit clauses $\{\overline{L}_1\}$ and $\{\overline{L}_2\}$ the truncation operator hypothesises L. In fact, we show in (Muggleton and Buntine, 1988) that it is sufficient for completeness to cover only the case in which τ_1 and τ_2 are empty substitutions, in which case L subsumes both $\overline{L}_1$ and $\overline{L}_2$. Moreover, without loss of completeness the operation can be made deterministic by constructing L to be the least general generalisation of $\overline{L}_1$ and $\overline{L}_2$. The details of this algorithm can be found in Plotkin (Plotkin, 1971). In CIGOL this operation has been extended to deal with arbitrarily large sets of unit clauses.

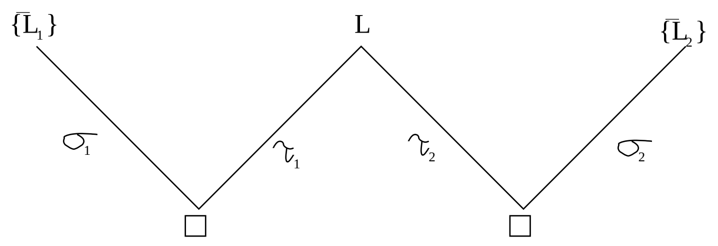

Figure 6: The truncation operator

5 CIGOL

The operators described in Section 4 have been implemented within the learning system CIGOL. CIGOL is an interactive Prolog program which incrementally constructs first-order Horn clause theories from example clauses presented by a human teacher. Questions are asked of the teacher in order to verify generalisations made by the truncation and absorption operators. CIGOL also asks the teacher to name new predicates introduced using the intra-construction operator. The following is a top-level description of CIGOL.

Algorithm 3 CIGOL

```
Let theory $T := \emptyset$ and counter-clauses $N := \emptyset$
Forever Do
  Let $I :=$ term-indexing(T)
  Read example clause E from teacher
  $E' :=$ best-agreed-truncation$(E, T, N, I)$
  inverse-prove$(\{E'\}, T, N, I)$
  reduce-pclauses$(T, N, I)$
Repeat
```

In the algorithm above, T is a set of Horn clauses representing the growing theory. The set of clauses N is composed of false clauses. These act as counter-examples of predicates and are added to when the teacher rejects generalisations posed within procedures best-agreed-truncation and inverse-prove. The procedure reduce-clauses removes redundant clauses using Buntine's (Buntine, 1986) redundancy algorithm.

The truncation operator is applied to new unit clause examples. The procedure best-agreed-truncation develops a subset T' of the unit clauses in T (indexed using I) such that E' is the least general generalisation of $T' \cup \{E\}$. T' is found using a best-first search based on minimising the size of $(T - T') \cup \{E'\}$. The size of syntactic objects is defined as follows.

size-of clause set $\{C_1, .., C_n\} = 1 + \sum(\text{size-of clause}C_i), \quad 1 \leq i \leq n$
size-of clause $\{L_1, .., L_n\} = 1 + \sum(\text{size-of literal}L_i), \quad 1 \leq i \leq n$
size-of literal or term $f(t_1, .., t_n) = 2 + \sum(\text{size-of}t_i), \quad 1 \leq i \leq n$
size-of variable $v = 1$

The search within best-agreed truncation halts at local minima. At this point CIGOL tests to see whether E' can be shown to be true or false using T and N. This is based on depth-bounded theorem proving, again using Buntine's redundancy algorithm. When it cannot be shown to be true or false the teacher is asked about the truth of E'. If the teacher rejects the generalisation then E' is added to N and the search resumes. Otherwise T becomes $(T - T') \cup \{E'\}$, I is updated and E' is returned by best-agreed-truncation.

The amount of search involved with best agreed truncation is reduced by use of the indexing I. Thus the search up to a local minimum for applying the truncation operator is carried out with upper-bound time complexity $O(|S'|^2)$ where S' is the number of unit clauses in T which have the same predicated symbol and arity as E.

The absorption and intra-construction operators are applied within inverse-prove. The inverse-prove procedure works in a way, which is similar to the goal reduction mechanism of Prolog. The difference is that whereas Prolog constructs resolution proofs of goal clauses from a given program, CIGOL uses inverse resolution to construct a resolution proof backward from the truncated examples. The clauses at the leaves of CIGOL's proof trees are then added to the background theory T. Clauses at the internal nodes of the proof tree are deleted from the theory, since these clauses are, by definition, implied by T. The structure of the inverse-prove procedure is given below.

Algorithm 4 *The inverse-prove procedure*

```
Input: Goal clauses Gs, theory T, counter-clauses N and term-indexing I
ForEach goal clause G in Gs
    NewGs := best-agreed-operator(G, T, N, I)
    inverse-prove(NewGs, T, N, I)
Repeat
```

The procedure best-agreed-operation works in the same way as best-agreed-truncation except that I is used to index into the terms and literals within T in order to search for absorption and intra-construction operators efficiently. In this way, the search up to a local minimum for applying the absorption operator is carried out with the upper-bound time complexity $O(U.V)$ where U is the number of subterms G and V is the number of terms within T which have corresponding function symbols and arity. The search up to a local minimum for applying the intra-construction operator is carried out with upper-bound time complexity $O(|T|.M.D)$ where $|T|$ is the number of clauses in T, M is the maximum arity of any term or literal in G and D is the maximum depth of the literals in G.

6 Discussion

Apart from the predicates learned in Sections 2.1 and 2.2, CIGOL has been capable of successfully learning a wide range of first-order concepts, each requiring only a small number of examples. The following are among the concepts learned so far:

list-reverse - invented auxiliary predicate "append"

list-minimum - minimum number in list, invented auxiliary "<" (see example 5)

ordered-binary-tree-insert - invented auxiliary "<"

merge-sort - given split and "<", invented "merge"

The reader will notice that in the last case a fair amount of background knowledge was necessary to learn the predicate. This seems to indicate a limit to the complexity of predicates which can be learned given only examples of the top-level predicate. It is not clear at this stage whether this limitation is fundamental to the approach, or whether it can be overcome by using a different control strategy. One hope is that non-incremental control would help. This would give the new version of CIGOL a control strategy similar to that of its propositional predecessor Duce. Indeed, Duce was capable of hierarchically decomposing problems on a much larger scale than anything CIGOL has been applied to.

As it is, CIGOL is capable of learning as wide a set of Prolog predicates as Shapiro's Model Inference System (MIS) (Shapiro, 1983) using far fewer examples. More importantly, systems such as MIS only carry out generalisation of underspecified predicates, without the invention of new auxiliary concepts.

It is widely believed that a first-order framework is too unconstrained for effective learning. However, we have shown here that just as resolution and unification can produce well-constrained deductive problem solvers such as Prolog, these constraints in reverse strongly limit the search for inductive generalisations (see polynomial complexity results in Section 5).

In conclusion, it is worth noting that choice of first-order clausal logic has several distinct advantages as a knowledge representation for machine learning. Predicate calculus' long history has left a particularly well-laid theoretical foundation. Moreover the notion of generality, central to all forms of machine learning, maps directly to the concept of implication, which is the mainstay of logic. The advent of resolution (Robinson, 1965) provided the only known sound and complete mechanism for computing logical implication. Our hope is that the methods based on inverse resolution will prove as powerful within the context of Machine Learning.

Acknowledgements

We would like to thank Professor Donald Michie for useful suggestions and unfailing encouragement. We also thank Claude Sammut, Tim Niblett, Mike Bain, Peter Mowforth and other members of the Turing Institute Machine Learning group for constructive and illuminating discussions. Stephen Muggleton would like to thank Thirza for her help with improving the style of certain sections of this paper. This work was supported by the British Government's Alvey Logic Database Demonstrator project.

References

Buntine, W. (1986). Generalised subsumption. In *Proc. of the 7th European Conference on Artificial Intelligence (ECAI-86)*. European Coordinating Committee for Artificial Intelligence.

Clocksin, W. and Mellish, C. (1981). *Programming in Prolog*. Springer-Verlag, Berlin.

DeJong, G. and Mooney, R. (1986). Explanation-based learning: an alternative view. *Machine Learning*, 1(2):145–176.

Epstein, S. (1987). On the discovery of mathematical theorems. In *IJCAI-87*, pages 194–197, Los Angeles, CA. Kaufmann.

Kedar-Cabelli, S. and McCarty, L. (1987). Explanation-based generalization as resolution theorem proving. In Langley, P., editor, *Proceedings of the Fourth International Workshop on Machine Learning*, pages 383–389, Los Altos. Morgan Kaufmann.

Langley, P., Bradshaw, G., and Simon, H. (1983). Rediscovering chemistry with the Bacon system. In Michalski, R., Carbonnel, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 307–330. Tioga, Palo Alto, CA.

Lassez, J.-L., Maher, M., and Marriot, K. (1986). Unification revisited. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, Los Altos,CA.

Lenat, D. (1981). On automated scientific theory formation: a case study using the AM program. In Hayes, J. and Michie, D., editors, *Machine Intelligence 9*. Horwood, New York.

Michalski, R. and Stepp, R. (1982). Revealing conceptual structure in data by inductive inference. In Hayes, J., Michie, D., and Pao, Y., editors, *Machine Intelligence 10*, pages 173–196. Ellis Horwood, Chichester, UK.

Mitchell, T., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80.

Muggleton, S. (1987). Duce, an oracle based approach to constructive induction. In *IJCAI-87*, pages 287–292. Kaufmann.

Muggleton, S. (1991). Inverting the resolution principle. In *Machine Intellience 12*, pages 93–104. Oxford University Press.

Muggleton, S. and Buntine, W. (1988). Towards constructive induction in first-order predicate calculus. TIRM 88-03, The Turing Institute, Glasgow.

Plotkin, G. (1971). *Automatic Methods of Inductive Inference.* PhD thesis, Edinburgh University.

Rendell, L. (1985). Substantial constructive induction using layered information compression: tractable feature formation in search. In *IJCAI-85*, pages 650–658. Kaufmann.

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

Sammut, C. (1981). *Learning concepts by performing experiments.* PhD thesis, Department of Computer Science, University of New South Wales, Australia.

Sammut, C. and Banerji, R. (1986). Learning concepts by asking questions. In Michalski, R., Carbonnel, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach. Vol. 2*, pages 167–192. Kaufmann, Los Altos, CA.

Shapiro, E. (1983). *Algorithmic program debugging.* MIT Press.

Utgoff, P. (1986). *Machine Learning of Inductive Bias.* Kluwer, Boston, MA.

Wirth, R. (1988). Completing proofs by inverting resolution. Technical report, University of Tuebingen, Tuebingen, West Germany.