New Generation Computing, 8 (1991) 295-318
OHMSHA, LTD. and Springer-Verlag



# Inductive Logic Programming

Stephen MUGGLETON
*The Turing Institute,*
*36 North Hanover St.,*
*Glasgow G1 2AD,*
*United Kingdom.*

Received 30 November 1990

**Abstract** A new research area, Inductive Logic Programming, is presently emerging. While inheriting various positive characteristics of the parent subjects of Logic Programming and Machine Learning, it is hoped that the new area will overcome many of the limitations of its forebears. The background to present developments within this area is discussed and various goals and aspirations for the increasing body of researchers are identified. Inductive Logic Programming needs to be based on sound principles from both Logic and Statistics. On the side of statistical justification of hypotheses we discuss the possible relationship between Algorithmic Complexity theory and Probably-Approximately-Correct (PAC) Learning. In terms of logic we provide a unifying framework for Muggleton and Buntine's Inverse Resolution (IR) and Plotkin's Relative Least General Generalisation (RLGG) by rederiving RLGG in terms of IR. This leads to a discussion of the feasibility of extending the RLGG framework to allow for the invention of new predicates, previously discussed only within the context of IR.

**Keywords:** Learning, logic programming, induction, predicate invention, inverse resolution, information compression

## 1 Introduction

Before evaluating the present developments in Inductive Logic Programming it is worth briefly reviewing the inter-related histories of deductive and inductive logic.

## 1.1 Deductive logic

Although logic has been studied since the time of Aristotle's *Prior Analytics* it was only during the 19th and early 20th centuries that theoreticians such as Boole and Frege transformed it into a rigorous mathematical science. During the present century a school of philosophers, known as logical positivists, have promoted the view that logic is the ultimate foundation not only of mathematics but of all sciences. According to this view, every mathematical statement can be phrased within the logical language of first order predicate calculus and all valid scientific reasoning is based on logical derivation from a set of pre-conceived axioms. Logical positivism was provided with powerful ammunition by Gödel's demonstration [9] that a small collection of sound rules of inference was complete for deriving all consequences of formulae in first order predicate calculus. Much later Robinson demonstrated [34] that a single rule of inference, called resolution, is both sound and complete for proving statements within this calculus (see Appendix A). For the purposes of applying resolution, formulae are normalised to what is known as *clausal form.* Robinson's discovery was of fundamental importance to the application of logic within computer science. Throughout the early 1970's Colmerauer and Kowalski [17] were instrumental in the development of the logic based programming language Prolog. Prolog statements are phrased in a restricted clausal form called Horn clause logic. All computations within Prolog take the form of logical proofs based on the application of the rule of resolution. Since its inception Prolog has developed into a widely used programming language [3, 41] and was chosen as the core language for Japan's Fifth Generation Project. Prolog's development has also spawned the rigorous theoretical school of Logic Programming [20].

## 1.2 Induction

Despite the self-evident success of logical deduction, a certain question has cropped up time and again throughout its development. If all human and computer reasoning proceeds from logical axioms, then where do logical axioms proceed from? The widely accepted answer to this is that logical axioms, representing generalised beliefs, can be constructed from particular facts using *inductive* reasoning. In turn, facts are derived from the senses. The history of inductive reasoning interacts with every development of its deductive counterpart. In ancient Greece inductive reasoning played a key role in Socrates' dialectic discussions. These are described in Plato's *The Last days of Socrates.* In these discussions concepts were developed and refined using a series of examples and counter-examples drawn from every day life. In the seventeenth century Bacon in the *Novum Organum* gave the first detailed description of the inductive scientific method. From the eighteenth century onwards, methods developed originally for predicting the outcome of games of chance developed into the mathematical discipline of Statistics. In turn, Statistics went on to have a central role in the evaluation of scientific hypotheses.

In 1931, a year after Gödel's previously mentioned completeness result, Gödel [12] published his more famous incompleteness theorem. According to this theorem Peano's axiomisation of arithmetic, and any first order theory containing it, is either

| | Application | No. of Rules | Development (Man-Yrs) | Maintenance (Man-Yrs/Yr) | Inductive Tools |
|---|---|---|---|---|---|
| MYCIN | medical diagnosis | 400 | 100 | N/A | N/A |
| XCON | VAX computer configuration | 8,000 | 180 | 30 | N/A |
| GASOIL | hydrocarbon separation system configuration | 2,800 | 1 | 0.1 | ExpertEase and Extran 7 |
| BMT | configuration of fire-protection equipment in buildings | 30,000 | 9 | 2.0 | 1st Class and RuleMaster |

Figure 1: Expert systems built with and without use of induction

self-contradictory or incomplete for the purposes of deriving certain arithmetic statements. This discovery prompted Turing to attempt to show [42] that problems concerning incompleteness of logical theories could be overcome by the use of an *oracle* capable of verifying underivable statements. Turing [43] later came to believe that Gödel's incompleteness theorem required that intelligent machines be capable of learning from examples.

Various logical positivists including Carnap [5] have developed statistical theories for confirming scientific hypotheses posed in first order logic. Although Plotkin [31] in the 1970's and Shapiro [37] in the 1980's worked on computer-based inductive systems within the framework of full first order logic, most successes within the field of Machine Learning have derived from systems which construct hypotheses within the limits of propositional logic. The major successes here have been in the area of inductive construction of expert systems (see [27]). The properties of various expert systems are given in Figure 1. The first two, MYCIN [38] and XCON [10] were built using hand-coding of rules. The second two, GASOIL [39] and BMT [15] were built using software derived from Quinlan's inductive decision tree building algorithm ID3 [32]. It should be noted that the inductively constructed BMT is by far the largest expert system in full-time commercial use. From the perspective of software engineering, it is also worth noting the sizeable reductions in development and maintenance times for inductively constructed systems. It seems likely that technologies based on inductive inference will play an increasingly important role in future software development.

### 1.3 Shortcomings of present inductive systems

Along with the successes of this technology, the following limitations are now becoming apparent.

- **Restricted representation.** Propositional level systems cannot be used in areas requiring essentially relational knowledge representations. These areas include temporal reasoning, scheduling, planning, qualitative reasoning, natural language and spatial reasoning. Problems also occur in other areas involving arbitrarily complex structural relationships such as prediction of protein folding and DNA gene mapping.
- **Inability to make use of background knowledge.** Human inductive reasoners make use of vast amounts of background knowledge when learning. Inductive algorithms such as ID3 use only a fixed set of attributes attached to each example. Explanation-based learning (EBL) [22] attempted to overcome this limitation by redefining the learning problem. In EBL hypotheses are constrained to being those derivable from background knowledge. Since background knowledge is rarely complete in applications, this constraint is now generally believed to be over-restrictive.
- **Strong bias of vocabulary.** Present inductive systems construct hypotheses within the limits of a fixed vocabulary of propositional attributes. An increasing amount of Machine Learning research [24, 28, 1, 44, 35, 16, 19] is concerned with algorithms capable of inventing auxiliary predicates when insufficient background knowledge is provided.

## 2 Inductive Logic Programming

A growing body of researchers have started to work on problems of inductive reasoning within the confines of pure Prolog. In this paper the phrase *Inductive Logic Programming* is used to describe this research area, formed at the intersection of Logic Programming and Machine Learning. In line with its parent subjects it would seem likely that the area will subdivide into three related strands of research: theory, implementation and experimental application. While model-theoretic semantics [20] and PAC-learning [8] theory are likely to be influential, neither seem totally adequate for the problems involved. The general goals of research for such an area should be to produce a widely used technology with a firm theoretical foundation. Ideally a uniform theory, such as that behind Logic Programming will emerge.

### 2.1 Past and present research

In his thesis Plotkin [31] laid the foundations for much of the present activity. Plotkin did not restrict himself to Horn clause logic. This is hardly surprising

given that logic programming had not yet come into existence. His major contributions were 1) the introduction of *relative subsumption*, a relationship of generality between clauses and 2) the inductive mechanism of *relative least general generalisation* (RLGG). Plotkin's main theoretical result was negative. This was that there is in general no finite relative least general generalisation of two clauses. His implementation of RLGG was thus severely restricted. This negative result of Plotkin's prompted Shapiro [37] to investigate an approach to Horn clause induction in which the search for hypotheses is from general to specific, rather than Plotkin's specific to general approach. Shapiro also investigated a technique called algorithmic debugging. Given a Prolog program which is either incomplete, incorrect or non-terminating, Shapiro's system diagnoses which clause is faulty and then replaces that clause.

Both Plotkin and Shapiro managed to prove that their learning systems could identify first order theories in the limit, according to Gold's definition [13]. Nowadays Valiant's definition of PAC-learning [14] (see Section 3.4) is generally agreed to provide a better approach to identification than Gold's methodology. However, PAC methods have largely been applied to propositional-level learning and have as yet to be demonstrated capable of dealing with inductive learning in full first order logic. It is also worth noting that despite the identification in the limit guarantees for Plotkin's and Shapiro's systems, both were very inefficient and were in practice only demonstrated on very limited problems.

Sammut and Banerji [36] describe a system called MARVIN which generalises a single example at a time with reference to a set of background clauses. At each stage a set of ground atoms $F$ representing the example are matched against the body of a background clause $H \leftarrow B$. The matching facts $B\theta$ are replaced by $H\theta$ within $F$, where $\theta$ is a ground substitution. This was the sole generalisation operation used. The method was demonstrated on a wide set of problems. Muggleton and Buntine [28] later showed that Sammut and Banerji's generalisation operation was a special case of inverting a step in a resolution proof. A set of more general constraints were derived, and two related operations were named the 'V' operators (see Section 4.1). Another pair of operators ('W'), also based on inverting resolution, were shown to be capable of "inventing" predicates which were not available within the background knowledge (Section 5.1). Banerji [1], Wirth [44] Ishizaka [16], Ling and Dawes [19] and Rouveirol and Puget [35] have also described related methods for "inventing" new predicates. However, although predicate invention has been demonstrated on large scale problems within a propositional setting [24, 25] this is not yet the case for any first order learning systems.

Recently Quinlan [33] has described a highly efficient program, called FOIL, which induces first order Horn clauses. The method relies on a general to specific heuristic search which is guided by an information criterion related to entropy. Quinlan sees his approach as being a natural extension of ID3 [32]. He notes that the search can be highly myopic, and is unable to learn predicates such as list-reversal and integer-multiplication.

Attempts have been made recently by Buntine [4], Frisch and Page [11] and Muggleton and Feng [29] to find ways around Plotkin's negative RLGG results.

| Domain | Clause | No. of examples | No. of atoms in background |
|---|---|---|---|
| Qualitative modelling (U-tube) | state(la:A/B,lb:C/D,fab:E/B,fba:F/D) ← deriv(la:A/B,fba:F/D), deriv(lb:C/D,fab:E/B), minus(la:A/B,lb:G/D,[]), minus(la:G/B,lb:C/D,[]). | 169 | 5408 |
| Temporal reasoning (satellite power supply) | bus_voltage(0,TimeA) ← asr_switch_1(Val,TimeA), relay_a011(0,TimeA), successor(TimeC,TimeA), insunlight(Val,TimeC). | 131 | 3584 |
| Protein folding | alpha(Prot,Pos) ← ... position(Prot,Pos,Res), hydrophobic(Res), less_hydrophobic(Rminus4,Res), less_hydrophobic(Rplus1,Res), ... | 1612 | 4185 |

Figure 2: Clauses built by Golem in real-world domains

Using a Horn clause framework Buntine tried restricting the model of the background knowledge to being a finite depth Herbrand model. The approach had limited success due to the complexity of constructed clauses. Frisch and Page have tried restricting the hypothesis language to being a strictly sorted logic with more success. Muggleton and Feng [29] apply a "determinate" restriction to hypotheses. By this it is meant that there is a unique choice of ground substitution for every resolution involved in the refutation of any ground goal using the hypothesised clause. Muggleton and Feng's learning program, Golem, has been demonstrated [29] to have a level of efficiency similar to Quinlan's FOIL but without the accompanying loss of scope. Although predicates such as list-reverse, integer-multiply and quick-sort were reported in [29], the following section provides some details of real world applications of Golem.

## 2.2 Real-world domains

A great deal of research has been invested in inductive logic programming algorithms. A general feeling is emerging that it is time to move on from the toy worlds of arch building and list and number theoretic predicates. Figure 2 shows various clauses developed as part of ongoing real-world application projects using Muggleton and Feng's [29] Golem. The first clause represents a qualitative model of a U-tube, developed in a collaborative project between the author and Ivan Bratko. The clause was built using Kuipers' [18] theory of qualitative reasoning

as background knowledge, encoded as 5408 ground atoms. The clause relates the levels of the two water surfaces, la and lb to the directions of flow fab and fba. Various qualitative relationships concerning the derivative of flow (deriv) and relationships between the signs (minus) of various qualitative values were discovered by Golem.

In the second clause, constructed by Feng for a satellite application, a simple temporal relationship was discovered. The clause predicts the bus voltage at Time A on the basis of various relay and switch values together with whether the satellite's solar panels are in sunlight. Simple temporal reasoning is achieved by the use of a time variable associated with each predicate.

The third clause was developed in collaborative work being carried out between the author and Ross King. In this clause a particular amino acid residue position (Pos) within a protein (Prot) is predicted to be part of a secondary structure known as an alpha-helix (alpha). The clause takes into account whether the residue (Res) at position Pos is hydrophobic, i.e. water rejecting. It also compares the hydrophobicity of Res to that of the residues at positions four before (Rminus4) and one after Res.

Each of the clauses in Table 2 were constructed in under 30 seconds on a SparcStation330, and each involved the use of relatively large amounts of foreground and background information.

The application areas of qualitative model construction, construction of temporal models and protein structure prediction are ideally suited to inductive logic programming techniques. However, the author believes these are only a foot in the door of a new world of untried application areas for induction.

# 3 Theoretical foundations

In this section we discuss issues concerning the theoretical foundations of inductive logic programming. Definitions from logic can be found in Appendix A. This section serves as a basis for the discussion of inverse resolution and relative least general generalisation within the following sections.

## 3.1 Generality

The relationship of generality between formulae can be defined as follows.

**Definition 1** *Formula $A$ is more general than formula $B$ if and only if $A \models B$ and $B \not\models A$.*

This definition imposes the lattice shown in Figure 3 on the set of all formulae. In Logic Programming terms the top of the lattice is the empty clause (or $\square$) and the bottom of the lattice is the empty program (or $\blacksquare$). Infinite ascending and descending chains exist within the lattice. Ascending chains can be constructed by conjoining formulae. Conversely, descending chains can be constructed by disjoining formulae.

The notion of relative generality of formulae will be used in the discussion of inductive inference in the next subsection. Relative generality is defined as follows.

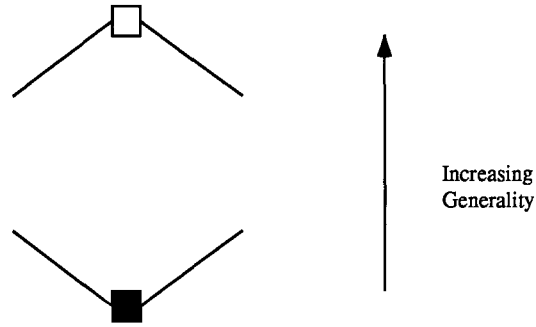


Figure 3: The generality lattice for formulae

**Definition 2** *Formula $A$ is more general than formula $B$ relative to formula $C$ if and only if $C \wedge A \models B$ and $C \wedge B \not\models A$.*

Formulae are partitioned into a set of equivalence classes using the following definition.

**Definition 3** *Formula $A$ is logically equivalent to formula $B$ if and only if $A \models B$ and $B \models A$.*

This definition suggests general-purpose methods for finding redundancy within formulae. Thus

**Definition 4** *Clause $C$ is redundant within logic program $P \wedge C$ if and only if $P \wedge C$ is logically equivalent to $P$.*

Logic programs can be reduced by removing all redundant clauses using theorem proving. Also

**Definition 5** *Literal $l$ is logically redundant within the clause $C \vee l$ in the logic program $P \wedge (C \vee l)$ if and only if $P \wedge (C \vee l)$ is logically equivalent to $P \wedge C$.*

Maher in [21] discusses a number of different notions of equivalence for logic programs including that of Definition 3. See also Buntine [4] and Niblett [30] for a detailed discussion of generality.

## 3.2 A general setting for induction

In the general inductive setting we are provided with three languages.

$\mathcal{L}_O$ : the language of observations
$\mathcal{L}_B$ : the language of background knowledge
$\mathcal{L}_H$ : the language of hypotheses

The general inductive problem is as follows: given a consistent set of examples or observations $O \subseteq \mathcal{L}_O$ and consistent background knowledge $B \subseteq \mathcal{L}_B$ find an hypothesis $H \in \mathcal{L}_H$ such that

$$B \wedge H \vdash O \tag{1}$$

As such, the problem is under-constrained. Usually, $\mathcal{L}_O$ is required to contain only ground literals. This still leaves a very unconstrained choice for $H$. Often $H$ is restricted to being a single clause. This is the case for Plotkin [31] and Shapiro [37] though not the case in Muggleton and Buntine [28] in which predicate invention introduces a set of clauses. $H$ can be constrained to be the most general hypothesis relative $B$ as in Shapiro [37] or the least general relative to $B$ as in Plotkin [31]. Alternatively, $H$ can be chosen to be that which produces the maximum information compression of $O$ relative to $B$ as in [26]. The choice of which constraint to apply is necessarily tied to our notions of justification of hypotheses.

## 3.3 Inductive inference and justification

As already stated, inductive inference involves the use of background knowledge to construct an hypothesis which agrees with some set of observations according to relationship (1). Clearly such a form of inference is not sound in the logical sense since $H$ does not necessarily follow from $B$ and $O$. In the worst case there could be an infinite number of contending hypotheses all of which fit the relationship shown above. For this reason we require some additional non-logical constraint to justify any particular hypothesis. In the 1950's Carnap [5] and others suggested "confirmation theories" aimed at providing a statistical underpinning to the problem of inductive inference. Various difficulties and paradoxes were encountered with these approaches which meant that they were never applied within machine learning programs [23].

## 3.4 PAC-learning

One popular machine learning approach to the problem of constructing highly probable hypotheses is the PAC (Probably Approximately Correct) model of learning proposed by Valiant [14]. According to Valiant's model a learning agent is not concerned with constructing an exact concept definition. Instead we choose a class of hypotheses that we would like to be able to deal with. We are then given a set of examples of the target concept. The cardinality of this example set must be at most a polynomial function of the size of the vocabulary used in constructing hypotheses. A class of hypotheses is PAC-learnable whenever we can guarantee with high probability that an arbitrarily chosen hypothesis which is consistent with the examples will agree with most subsequently presented examples. This has been shown to be possible when various constraints are placed on the hypothesis language. For instance, one such constraint involves placing a constant bound $k$ on the allowable size of conjunctions within a boolean DNF concept description (called k-DNF). In

a recent paper Dietterich [7] showed that for the purposes of learning DNF propositional descriptions the class of PAC-learnable concepts is highly restricted. There are $2^{2^m}$ different boolean functions of $m$ input values. However, Dietterich shows on the basis of general Valiant-learnability results due to Ehrenfeucht, Haussler et al [8] that given $n$ examples at most $O(2^{nm})$ of these functions are PAC-learnable. Dietterich [7] expresses concern over these results since it would seem that inductive learning algorithms must be restricted to searching highly constrained hypothesis spaces. This might be taken to suggest that it is not worth straying from propositional logic as a hypothesis language, since it is not even possible to PAC-learn the whole class of propositional descriptions.

## 3.5 Information compression

The following observation defeats this line of argument. Given $n$ examples of a concept, consider the class of boolean hypotheses which imply these $n$ examples, but whose minimal DNF description length is less than the description length of the examples. Since each example is defined in terms of $m$ input values and one output value, the description length of the examples will be $n(m+1)$ bits. However, at most $2^{n(m+1)}$ different hypotheses are describable in $n(m+1)$ bits. Note the close similarity between this upper bound and that given by Dietterich. This seems to provide at least strong circumstantial evidence for the belief that the class of PAC-learnable hypotheses is the same as the class of hypotheses which are textually less complex than the examples on which they are based. For the purposes of reference we will call this the PAC-Information-Compression-Equivalence (PICE) conjecture. This conjecture, if true would have a number of desirable side-effects. Firstly, the textual simplicity of examples relative to a given hypothesis is easy to measure. Secondly, it is relatively straightforward to interpret textual compression within any formalism, be it propositional logic, first-order logic or context-free grammar rules. Thirdly, this would bring the PAC-learning model into line with other approaches developed from algorithmic complexity theory [40, 6, 2] and Bayesian statistics. According to Bayes' Law

$$Pr(H|O) = \frac{Pr(H).Pr(O|H)}{Pr(O)}$$

If we treat bit-encoded descriptions as though they were the outcome of repeatedly tossing an unbiased coin, the prior probability of a string of bits $s$ of length $l$ is simply $2^{-l}$. Now by taking logs we can rewrite Bayes' Law in information terms as follows.

$$I(H|O) = I(H) + I(O|H) - I(O)$$

where $I(X)$ represents the minimal encoding length of $X$. For the probability $Pr(H|O)$ to approach 1, $I(H|O)$ must approach 0, in which case $I(H) + I(O|H)$ must be at least as great and approximately equal to $I(O)$. Although the minimal

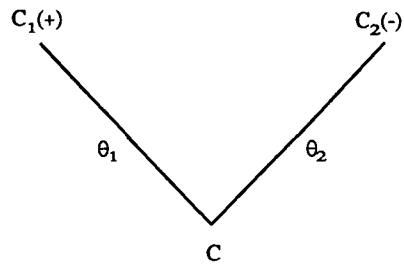


Figure 4: Single resolution

encodings $I(H)$ and $I(O)$ have a clear interpretation, one might ask what the meaning of $I(O|H)$ should be in terms of logic. The natural interpretation that we choose here is that $I(O|H)$ is the minimal encoding of the derivation or proof of $O$ from $H$. This interpretation has been shown to give good results elsewhere [26]. Note that this interpretation also has the desirable side-effect of discriminating against hypotheses which require very long proofs to derive the examples on which they are based, thus nicely combining the notions of space and time complexity of hypotheses in terms of a single information measure.

# 4 The relationship between IR and RLGG

In this section we review and extend Muggleton and Buntine's [28] Inverse Resolution (IR) techniques leading to a unified framework for IR and Plotkin's RLGG.

## 4.1 The 'V' operators

Figure 4 is a diagrammatic representation of a resolution step. Resolution derives the clause at the base of the 'V' given the two clauses on the arms. In contrast, a 'V' operator derives one of the clauses on the arm 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 logic program containing $C_1$ and $C_2$, $C$ is redundant according to Definition 4. Other clauses within the background theory may also be made redundant by the addition of $C_1$ or $C_2$ to the growing theory. It is the ability to discard redundant clauses that allows these operators to simplify and compact theories (see section 3.5).

The *absorption* operator constructs $C_2$ given $C_1$ and $C$. Conversely, the construction of $C_1$ from $C_2$ and $C$ is called the *identification* operator. Together these operators are called the 'V' operators.

In [28] the 'V' operators are derived as a set of constraints from the following equation of resolution (see section A.3).

$$C = (C_1 - \{l_1\})\theta_1 \cup (C_2 - \{l_2\})\theta_2 \tag{2}$$

where $l_1$ is a positive literal in $C_1$, $l_2$ is a negative literal in $C_2$ and $\theta_1\theta_2$ is the mgu of $\neg l_1$ and $l_2$. Muggleton and Buntine [28] base the algebraic manipulation required to produce constraint equations for the 'V' operators on a number of assumptions. Among these is the assumption that the clauses $(C_1 - \{l_1\})\theta_1$ and $(C_2 - \{l_2\})\theta_2$ contain no common literals. This they call the *separability* assumption. The separability assumption will be avoided in this paper. Instead we develop the constraints necessary for most specific 'V' operators.

## 4.2 Most specific 'V' operators

With reference to equation (2), let $C_1'$ be $(C_1 - \{l_1\})\theta_1$ and $C_2'$ be $(C_2 - \{l_2\})\theta_2$. Thus $C = C_1' \cup C_2'$. Also let $D = C_1' - C_2'$. Thus $C_2' = C - D$ or rather $(C_2 - \{l_2\})\theta_2 = C - D$. Rewriting this we get

$$C_2 = (C - D)\theta_2^{-1} \cup \{l_2\} \tag{3}$$

Now, since $\theta_1\theta_2$ is the mgu of $\neg l_1$ and $l_2$, we know that $\neg l_1\theta_1 = l_2\theta_2$ and thus

$$l_2 = \neg l_1\theta_1\theta_2^{-1} \tag{4}$$

Substituting (4) into (3) we get

$$\begin{aligned} C_2 &= (C - D)\theta_2^{-1} \cup \{\neg l_1\}\theta_1\theta_2^{-1} \\ &= ((C - D) \cup \{\neg l_1\}\theta_1)\theta_2^{-1} \end{aligned} \tag{5}$$

For Horn clauses the choice of $l_1$ is unique since it is a positive literal, and is therefore the head of $C_1$. However, it should be noted that given $C$ and $C_1$ equation (5) represents a whole range of possible solutions. These depend on the choice of $D$, $\emptyset \subseteq D \subseteq C_1'$, the choice of $\theta_1$ and the choice of $\theta_2^{-1}$. However, given $\theta_1$ and $\theta_2^{-1}$ there is a unique most specific solution for $C_2$. Thus with $D = \emptyset$ we get

$$C_2 = (C \cup \{\neg l_1\}\theta_1)\theta_2^{-1})$$

Again there is a range of solutions dependent on our choice of $\theta_1$ and $\theta_2^{-1}$. However, if we are interested only in most specific solutions we can choose $\theta_2^{-1}$ to be empty as well. Any other assignment produces a more general clause. Using $C_2\downarrow$ to denote the most specific solution for the absorption operator we get

$$C_2\downarrow = (C \cup \{\neg l_1\}\theta_1)$$

Note that $\theta_1$ can be partly determined from $C$ and $C_1$. From equation (2) we can see that $C \supseteq (C_1 - \{l_1\}\theta_1)$, i.e. $C$ $\theta$-subsumes $(C - \{l_1\})$ with substitution $\theta_1$. Thus a substitution $\theta'_1 \subseteq \theta_1$ can be constructed by matching literals within $C$ and $C_1$. However, what should be done with variables in $l_1$ which are not in the domain of $\theta'_1$? In fact, we are free to apply any substitution to these variables, since $\theta_1$ is part of the mgu of $\neg l_1$ and $l_2$ and we are not given $l_2$. The problem can be avoided by assuming that every variable found in $l_1$ is found in some other literal in $C_1$. In this case $\theta'_1 = \theta_1$ since $\theta'_1$ is constructed by matching all literals within $(C_1 - \{l_1\})$ with literals in $C$. The constraint that all variables in the head $(l_1)$ of a background clause $(C_1)$ be found within the body of the clause $(C_1 - \{l\})$ is used by Muggleton and Feng [29] for construction of RLGG's. They call a logic program containing only such clauses syntactically generative and prove that all atoms derivable using resolution are ground. If we apply the stronger constraint that every variable in every background clause must be found in at least two literals we can, using the same reasoning as above, show that the most specific solution for the identification operator is

$$C_1\downarrow \;=\; (C \cup \{\neg l_2\}\theta_2) \tag{6}$$

For the purposes of this paper we use the phrase *weakly generative* in place of Muggleton and Feng's [29] *syntactically generative*. The phrase *strongly generative* is used to describe sets of clauses in which every variable in every clause is found in at least two literals of that clause. Note that for every application of the unrestricted absorption operator $C_2$ $\theta$-subsumes $C_2\downarrow$ with substitution $\theta_2$. Similarly for every application of the unrestricted identification operator $C_1$ $\theta$-subsumes $C_1\downarrow$ with substitution $\theta_1$.

## 4.3 The function $\mathcal{V}^n$

We now give a function which describes the set of most specific clauses that can be constructed by iterative application of the 'V' operators. The function $\mathcal{V}^n(P, C)$ : $GenerativeLogicPrograms \times Clauses \rightarrow ClauseTheories$ is defined recursively in a manner analogous to Robinson's function $\mathcal{R}^n(P)$

$$\begin{aligned}
&\mathcal{V}^0(P, C) = \{C\}\\
&\mathcal{V}^n(P, C) = \mathcal{V}^{n-1}(P, C) \cup \{(C' \cup \{\neg l\}\theta) : l \in C_{1,2} \in P \text{ and}\\
&\qquad\qquad C' \in \mathcal{V}^{n-1}(P, C) \text{ and } (C_{1,2} - \{l\})\theta \subseteq C'\}
\end{aligned}$$

The closure $\mathcal{V}^*(P, C)$ is simply $(\mathcal{V}^1(P, C) \cup \mathcal{V}^2(P, C) \cup \ldots)$. Note that $\mathcal{V}^n(P, C)$ contains not only all the most specific absorptions but also all the most specific identifications to depth $n$. In fact, the definition of $\mathcal{V}^n$ does not require that $P$ be a logic program. The definition works equally well when $P$ is an arbitrary set of strongly generative clauses.

**Theorem 6 Completeness of $\mathcal{V}^n$ wrt. linear derivation.** *Let $\{D_1, \ldots, D_{k-1}\}$ be a subset of the clauses in the strongly generative logic program $P$. Let $C$ be the resolvent of the linear derivation $(D_1 \cdot (D_2 \cdot \ldots (D_{k-1} \cdot C_k)\ldots))$ where $(X \cdot Y)$*

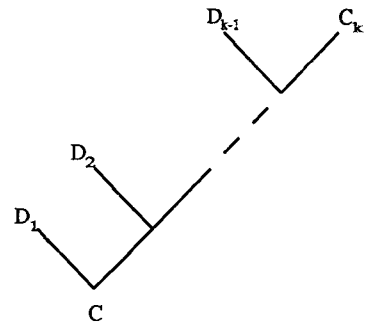


Figure 5: Linear derivation

*represents a resolvent of the clauses $X$ and $Y$ and $C_k \notin P$. For every such linear derivation $\mathcal{V}^k(P, C)$ contains a clause $C_k\downarrow$ such that $C_k$ $\theta$-subsumes $C_k\downarrow$.*
**Proof.** *Proof by induction on $k$. Trivial for $k = 0$. Assume $k = i$ and prove for $k = i+1$. By the definition and inductive assumption $\mathcal{V}^i(P, C)$ contains $C_i\downarrow$ which is $\theta$-subsumed by $C_i$. Let $C = (D_1 \cdot (D_2 \cdot \ldots (D_i \cdot C_{i+1}) \ldots))$ and $C_i = D_i \cdot C_{i+1}$ where $D_i \in P$, $l$ is the literal resolved on in $D_i$ and $\mu$ is the mgu in this resolution. By definition $\mathcal{V}^{i+1}(P, C)$ contains $C_{i+1}\downarrow = C_i\downarrow \cup \{\neg l\}\theta$. Due to the fact that $P$ is strongly generative the domain of $\theta$ is vars($l$). Since $(C_{i+1} - \{\neg l\})$ $\theta$-subsumes the resolvent $C_i$ with substitution $\mu$ it follows that $C_{i+1}$ $\theta$-subsumes $(C_i \cup \{\neg l\}\theta)$ and therefore $C_{i+1}$ $\theta$-subsumes $C_{i+1}\downarrow$.* □

Figure 5 illustrates Theorem 6. The depicted linear derivation of $C$ represents $(D_1 \cdot (D_2 \cdot \ldots (D_{k-1} \cdot C_k) \ldots))$.

As a corollary to Theorem 6 we note that the clause formed by taking the union of clauses in $\mathcal{V}^n(P, C)$ is subsumed by every clause constructed by iterative application of 'V' operators to $C$ and members of $P$.

**Corollary 7 Most specific inverse linear derivation.** *Let $\{D_1, \ldots, D_{k-1}\}$ be a subset of the clauses in the strongly generative logic program $P$. Let $C = (D_1 \cdot (D_2 \cdot \ldots (D_{k-1} \cdot C_k) \ldots))$ and $C_k \notin P$. For every such linear derivation $C_k$ $\theta$-subsumes $\bigcup \mathcal{V}^k(P, C)$.*
**Proof.** *Follows trivially from Theorem 6.*

## 4.4 Lgg of inverse linear derivations

Suppose we are given two clause, $C$ and $D$ and told that there is a clause $C_k \notin P$ such that both $C$ and $D$ have a linear derivation of depth at most $k$ from $C_k$ and $P$. From Corollary 7 we know that $C_k$ $\theta$-subsumes both $\bigcup \mathcal{V}^k(P, C)$ and $\bigcup \mathcal{V}^k(P, D)$.

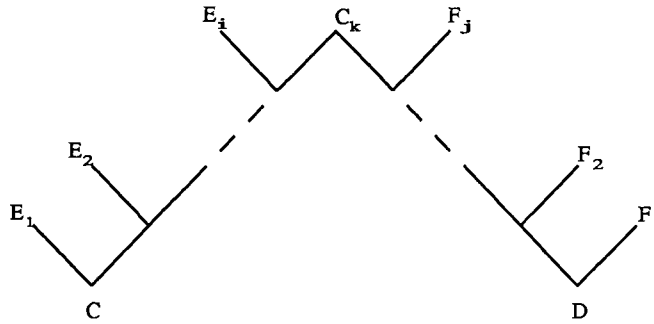


Figure 6: Inverse linear derivations of $C$ and $D$ with common $C_k$

Plotkin [31] investigated the lattice that $\theta$-subsumption defines over the set of all clauses. He defines the least general generalisation (lgg) $C$ of two clauses $D_1$ and $D_2$ as being the greatest lower bound within the clause lattice induced by $\theta$-subsumption and describes the following method for constructing lggs. The lgg of the terms $f(s_1,..,s_n)$ and $f(t_1,..,t_n)$ is $f(lgg(s_1,t_1),..,lgg(s_n,t_n))$. The lgg of the terms $f(s_1,..,s_n)$ and $g(t_1,..,t_n)$ where $f \neq g$ is the variable $v$ where $v$ represents this pair of terms throughout. The lgg of two atoms $p(s_1,..,s_n)$ and $p(t_1,..,t_n)$ is $p(lgg(s_1,t_1),..,lgg(s_n,t_n))$, the lgg being undefined when the sign or predicate symbols are unequal. The lgg of two clauses $C_1$ and $C_2$ is $\{l : l_1 \in C_1$ and $l_2 \in C_2$ and $l_1$ has the same sign and predicate symbol as $l_2$ and $l = lgg(l_1,l_2)\}$.

Thus the least general solution for $C_k$ is simply the least general generalisation of $\bigcup \mathcal{V}^k(P,C)$ and $\bigcup \mathcal{V}^k(P,D)$.

**Theorem 8 Lgg of inverse linear derivations.** *Let* $\{E_1,\ldots,E_i\}$ *and* $\{F_1,\ldots,F_j\}$ *be subsets of the strongly generative logic program* $P$. *Let* $C = (E_1 \cdot (E_2 \cdot \ldots (E_i \cdot C_k)\ldots))$ *and* $D = (F_1 \cdot (F_2 \cdot \ldots (F_j \cdot C_k)\ldots))$ *where* $C_k \notin P$, $k \geq i$ *and* $k \geq j$. *For every such pair of derivations* $C_k$ $\theta$-*subsumes* $lgg(\bigcup \mathcal{V}^k(P,C), \bigcup \mathcal{V}^k(P,D))$.
**Proof.** *Follows from Corollary 7 and Plotkin's definition of lgg.* □

Figure 6 illustrates the linear derivations of $C$ and $D$ from the common clause $C_k$.

**Example 9** *The following is a simple example illustrating the construction of the lgg of two inverse linear derivations.*

| | | | |
|---|---|---|---|
| *Let* | $C_2$ | = | *has_wings(X)* ← *bird(X)* |
| | $C_2'$ | = | *has_beak(X)* ← *bird(X)* |
| | $P$ | = | $\{C_2, C_2'\}$ |
| | $C$ | = | *has_wings(penguin)* ← |
| | $D$ | = | *has_beak(penguin)* ← |

$k = 1$
*Thus* $\mathcal{V}^0(P,C) = \{has\_wings(penguin) \leftarrow\}$

*Applying equation (6) to $C$ and $C_2$, with $l_2 = \neg bird(X)$ and $\theta_2 = \{X/penguin\}$ we get*

$$\bigcup\mathcal{V}^1(P,C) = has\_wings(penguin),\ bird(penguin) \leftarrow$$

*Applying equation (6) to $D$ and $C_2'$, with $l_2 = \neg bird(X)$ and $\theta_2 = \{X/penguin\}$ we get*

$$\bigcup\mathcal{V}^1(P,D) = has\_beak(penguin),\ bird(penguin) \leftarrow$$

*$C_k$ $\theta$-subsumes $lgg(\bigcup\mathcal{V}^1(P,C),\bigcup\mathcal{V}^1(P,D)) = bird(penguin) \leftarrow$*
*Letting $E_1 = C_2$ and $F_1 = C_2'$ we get $C = (E_1 \cdot C_k)$ and $D = (F_1 \cdot C_k)$ as expected.*

From the example it should be noted that $\bigcup\mathcal{V}^k(P,C)$ and $\text{lgg}(\bigcup\mathcal{V}^k(P,C),\bigcup\mathcal{V}^k(P,D))$ are not necessarily Horn clauses.

## 4.5 Relative least general generalisation

In this subsection we describe the relationship between Plotkin's RLGG and the lgg of inverse linear derivations of the previous subsections. Let $P$ be a logic program and $C$ and $D$ be two Horn clauses. The relative least general generalisation of $C$ and $D$, $rlgg_P(C,D)$ is the least general clause within the $\theta$-subsumption lattice for which $P \wedge rlgg_P(C,D) \vdash C \wedge D$ where $rlgg_P(C,D)$ is used only once in the derivation of $C$ and $D$.

This definition corresponds almost entirely to $\text{lgg}(\bigcup\mathcal{V}^k(P,C),\bigcup\mathcal{V}^k(P,D))$. The only differences are that $\bigcup\mathcal{V}^k(P,C)$ and $\bigcup\mathcal{V}^k(P,D)$ only take account of linear derivations of $C$ and $D$ with respect to $P$ and the depth of derivation is limited to $k$. However it is well known that linear derivation is sound and complete. In the case of the depth parameter $k$, Muggleton and Feng [29] describe a method for constructing RLGG's that uses a parameter $h$ in the same way as $k$ is used here.

**Theorem 10 Equivalence of RLGG and lgg of inverse linear derivations.** *Let $P$ be a logic program and let $C$ and $D$ be Horn clauses. $rlgg_P(C,D) = lgg(\bigcup\mathcal{V}^*(P,C),\bigcup\mathcal{V}^*(P,D))$.*
**Proof.** *The theorem follows from the definitions of $rlgg_P(C,D)$ and $lgg(\bigcup\mathcal{V}^*(P,C)$, $\bigcup\mathcal{V}^*(P,D))$.* □

Obviously, the clause $lgg(\bigcup\mathcal{V}^*(P,C),\bigcup\mathcal{V}^*(P,D))$ can be infinite. Plotkin [31] showed that $rlgg_P(C,D)$ can be infinite even when logically reduced. Note however that whereas Plotkin intended the reduced RLGG to be the hypothesis, within Muggleton and Buntine's [28] IR framework the most compact consistent clause which subsumes $lgg(\bigcup\mathcal{V}^k(P,C),\bigcup\mathcal{V}^k(P,D))$ will be chosen. A similar approach is found to be essential within Muggleton and Feng [29], since RLGG clauses can be very large, even when reduced using Definition 5.

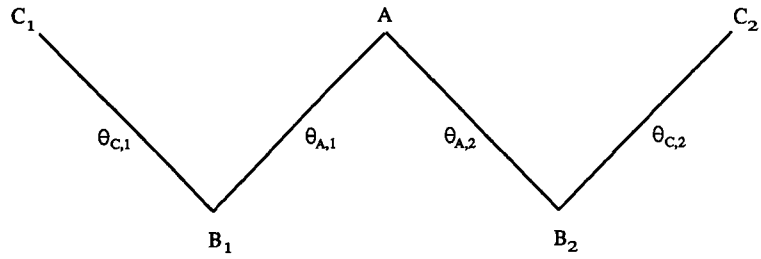


Figure 7: Two resolution steps with common clause A

# 5 Predicate invention

In the last section we described a unified framework covering both RLGG and IR. However, in Muggleton and Buntine's [28] description of IR a salient feature was the invention of new predicates. Predicate invention within the IR framework is carried out using the 'W' operators. In this section we review 'W' operators and discuss a method of incorporating a generalised 'W' operator into the combined IR-RLGG framework.

## 5.1 The 'W' operators

By combining together two resolution 'V's back-to-back we get a 'W' of the form shown in Figure 7. Assume that $C_1$ and $C_2$ resolve on a common literal $l$ within $A$ to produce $B_1$ and $B_2$ respectively. The 'W' operators construct the clauses $A$, $C_1$ and $C_2$ given $B_1$ and $B_2$. In the *intra-construction* operator $l$ is assumed negative while in the *inter-construction* operator $l$ is assumed 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 sense that a new predicate is *invented* by the 'W' operators. From equation (2) we get

$$B_i = (A - \{l\})\theta_{A,i} \cup (C_i - \{l_i\})\theta_{C,i}$$

where $i \in \{1,2\}$ and $\theta_{A,i}\theta_{C,i}$ is the mgu of $\neg l$ and $l_i$. Thus $(A - \{l\})$ $\theta$-subsumes each clause $B_i$. Clearly one possible solution is that $(A - \{l\}) = lgg(B_1, B_2)$. With this choice $\theta_{A,i}$ can be constructed by matching $(A - \{l\})$ with the corresponding literals of $B_i$. Muggleton and Buntine make the assumption that each $C_i$ is a unit clause, i.e.

$$C_i = \{l_i\}$$

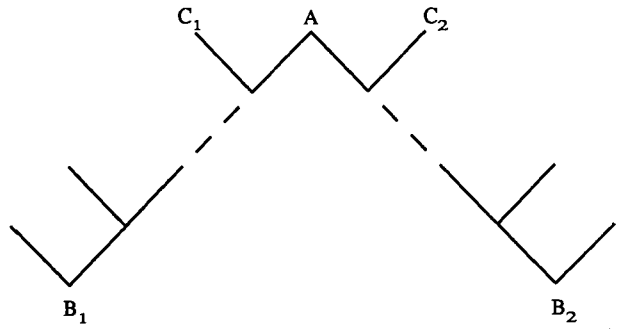


Figure 8: Generalised 'W' operator

and thus

$$B_i = (A - \{l\})\theta_{A,i}$$

They then construct a literal containing the variables in $\mathrm{dom}(\theta_{A,i})$. Assuming $\theta_{C,i}$ is empty gives

$$C_i = \{\neg l\}\theta_{A,i}$$

and

$$A = lgg(B_1, B_2) \cup \{l\} \tag{7}$$

## 5.2 Generalised 'W' operators

A more general form for the 'W' operators is shown in Figure 8. In this case 'W' forms the crown of a pair of inverse derivations of $B_1$ and $B_2$. Note the similarity to Figure 6. Equation (7) thus becomes

$$A = rlgg_P(B_1, B_2) \cup \{l\}$$

As in the intra-construction operator we assume that $l$ is negative. The most specific solutions for $\theta_{A,i}$ will be found as a bi-product of the formation of $rlgg_P(B_1,B_2)$. Thus the only choice is that of the variables and terms from $rlgg_P(B_1, B_2)$ which will appear in $l$. This is not an easy choice since rlgg's can contain many literals, terms and variables. We leave this as an open problem for further investigation.

# 6 Conclusion

Interest in inductive logic programming has increased substantially in the last few years. There now seems to be a new technology emerging which is capable of dealing with a wide number of application areas (see Section 2.2). Such a technology requires a uniform underlying theory in order to be successful in the same way as Logic Programming. The discovery of a common framework for IR and RLGG (section 4.5) is one step in this direction. Surprisingly both approaches demand that the logic programs involved be *generative* (see [29] and Section 4.2), but for independent reasons. It is rather intriguing that IR and RLGG should converge in this way since RLGG was based initially on the idea of inverting unification while IR is based on inverting resolution.

The IR-RLGG framework has a distinct advantage over the use of the 'V' operator in Muggleton and Buntine [28]. Whereas Muggleton and Buntine's method requires heuristic search to choose which 'V' operator to apply, the new approach constructs a unique solution for multiple 'V' operations without recourse to search. Other systems such as Quinlan's FOIL [33] also use heuristic search. The disadvantages of heuristic search are not only that it can be inefficient, but also that solutions can be missed due to local minima in the search space. These problems are avoided within the new IR-RLGG framework.

Rouveirol and Puget [35] describe an operator called saturation which carries out multiple 'V' operations. There is a clear relationship between the saturation operator and the function $\mathcal{V}^n$ (Section 4.3). The definition of $\cup\mathcal{V}^n$ can be seen as a formalisation of the saturation operator. Also Wirth [44] describes a method in which he takes the lgg of a number of 'V' operations. Wirth's method is a special case of $\text{lgg}(\cup\mathcal{V}^1(P,C),\cup\mathcal{V}^1(P,D))$.

Although the generalised 'W' method described in Section 5.2 seems promising, it needs much further investigation. For instance it might seem natural to devise a $\mathcal{W}^n$ operator along the lines of the $\mathcal{V}^n$ function. Clearly we are in need of a stronger formal framework for predicate invention. This should be tightly coupled with the IR-RLGG framework.

Finally, the author believes that work should be done on applying PAC-learning theory to inductive logic programming. A possible starting point might be the PICE conjecture of Section 3.5.

With the promise of solid theoretical foundations to come, Inductive Logic Programming has the potential of a very exciting future.

**Acknowledgements.** The author would like to thank Wray Buntine, Cao Feng, Michael Bain and Donald Michie for their helpful discussions and advice. This work was carried out at the Turing Institute and was supported by the Esprit Basic Research Action project ECOLES and the IED's Temporal Databases and Planning project. The author is presently supported by an SERC post-doctoral fellowship.

# A Definitions from logic

## A.1 Formulae in first order predicate calculus

A variable is represented by an upper case letter followed by a string of lower case letters and digits. A function symbol is a lower case letter followed by a string of lower case letters and digits. A predicate symbol is a lower case letter followed by a string of lower case letter and digits. The negation symbol is: $\neg$. A variable is a term, and a function symbol immediately followed by a bracketed n-tuple of terms is a term. Thus $f(g(X), h)$ is a term when $f$, $g$ and $h$ are function symbols and $X$ is a variable. A predicate symbol immediately followed by a bracketted n-tuple of terms is called an atomic formula. Both $A$ and $\neg A$ are literals whenever $A$ is an atomic formula. In this case $A$ is called a positive literal and $\neg A$ is called a negative literal. The literals $A$ and $\neg A$ are said to be each others complements and form, in either order, a complementary pair. A finite set (possibly empty) of literals is called a clause. The empty clause is represented by $\square$. A clause represents the disjunction of its literals. Thus the clause $\{A_1, A_2, ..\neg A_i, \neg A_{i+1}, ...\}$ can be equivalently represented as $(A_1 \vee A_2 \vee ..\neg A_i \vee \neg A_{i+1} \vee ...)$ or $A_1, A_2, .. \leftarrow A_i, A_{i+1}, ....$ A Horn clause is a clause which contains exactly one positive literal. The positive literal in a Horn clause is called the head of the clause while the negative literals are collectively called the body of the clause. A set of clauses is called a clausal theory. The empty clausal theory is represented by $\blacksquare$. A clausal theory represents the conjunction of its clauses. Thus the clausal theory $\{C_1, C_2, ...\}$ can be equivalently represented as $(C_1 \wedge C_2 \wedge ...)$. A set of Horn clauses is called a logic program. Apart from representing the empty clause and the empty theory, the symbols $\square$ and $\blacksquare$ represent the logical constants *False* and *True* respectively. Literals, clauses and clausal theories are all well-formed-formulae. Let $E$ be a wff or term. vars($E$) denotes the set of variables in $E$. $E$ is said to be ground if and only if vars($E$) $= \emptyset$.

## A.2 Models and substitutions

A set of ground literals which does not contain a complementary pair is called a model. Let $M$ be a model, $C$ be a clause and $\mathcal{C}$ be the set of all ground clauses obtained by replacing the variables in $C$ by ground terms. We say that $M$ is a model of $C$ if and only if each clause in $\mathcal{C}$ contains at least one literal found in $M$. $M$ is a model for clausal theory $T$ if and only if $M$ is a model for each clause in $T$. Let $F_1$ and $F_2$ be two wff's. We say that $F_1$ semantically entails $F_2$, or $F_1 \models F_2$ if and only if every model of $F_1$ is a model of $F_2$. We say that $F_1$ syntactically entails $F_2$ using $I$, or $F_1 \vdash_I F_2$, if and only if $F_2$ can be derived from $F_1$ using the set of deductive inference rules $I$. The set of inference rules $I$ is said to be deductively sound and complete if and only if $F_1 \vdash_I F_2$ whenever $F_1 \models F_2$. In this case we can drop the subscript and merely write $F_1 \vdash F_2$. Let $F_1$ and $F_2$ be two wff's. We say that $F_1$ is more general than $F_2$ if and only if $F_1 \vdash F_2$. A wff $F$ is satisfiable if there is a model for $F$ and unsatisfiable otherwise. $F$ is unsatisfiable if and only if $F \models \square$.

Let $\theta = \{v_1/t_1, .., v_n/t_n\}$. $\theta$ is said to be a substitution when each $v_i$ is a variable and each $t_i$ is a term, and for no distinct $i$ and $j$ is $v_i$ the same as $v_j$. The set $\{v_1, .., v_n\}$ is called the domain of $\theta$, or dom($\theta$), and $\{t_1, .., t_n\}$ the range of $\theta$, or rng($\theta$). We use lower case Greek letters to denote substitutions. Let $E$ be a well-formed formula or a term and $\theta = \{v_1/t_1, .., v_n/t_n\}$ be a substitution. The instantiation of $E$ by $\theta$, written $E\theta$, is formed by replacing every occurrence of $v_i$ in $E$ by $t_i$. Every sub-term within a given term or literal $W$ can be uniquely referenced by its *place* within $W$. Places within terms or literals are denoted by n-tuples of natural numbers and defined recursively as follows. The term at place $\langle i \rangle$ within $f(t_0, .., t_m)$ is $t_i$. The term at place $\langle i_0, .., i_n \rangle$ within $f(t_0, .., t_m)$ is the term at place $\langle i_1, .., i_n \rangle$ in $t_{i_0}$. Let $t$ be a term found at place $p$ in literal $L$, where $L$ is a literal within clause $C$. The place of $t$ in $C$ is denoted by the pair $\langle L, p \rangle$. Let $E$ be a clause or a term and $\theta = \{v_1/t_1, .., v_n/t_n\}$ be a substitution. The corresponding inverse substitution $\theta^{-1}$ is $\{(t_1, \{p_{1,1}, .., p_{1,m_1}\})/v_1, .., \langle t_n, \{p_{n,1}, .., p_{1,m_n}\}\rangle/v_n\}$. An inverse substitution is applied by replacing all $t_i$ at places $p_{i,1}, .., p_{i,m_i}$ within $E$ by $v_i$. Clearly $E\theta\theta^{-1} = E$. Note that an inverse substitution is not strictly a substitution but rather a rewrite. Let $C$ and $D$ be clauses. We say that $C$ $\theta$-subsumes $D$ if and only if there exists a substitution $\theta$ such that $C\theta \subseteq D$.

## A.3 Resolution

Let $F_1$ and $F_2$ be two wff's and $\theta$ be the substitution $\{u_1/v_1, ..u_n/v_n\}$ in which for every distinct $i$ and $j$, $v_i$ is different from $v_j$. $F_1\theta$ and $F_2\theta$ are said to be standardised apart whenever there is no variable which occurs in both $F_1\theta$ and $F_2\theta$. The substitution $\theta$ is said to be the unifier of the atoms $A$ and $A'$ whenever $A\theta = A'\theta$. $\mu$ is the most general unifier (mgu) of $A$ and $A'$ if and only if for all unifiers $\gamma$ of $A$ and $A'$ there exists a substitution $\delta$ such that $(A\mu)\delta = A\gamma$. $((C - \{A\}) \cup (D - \{\neg A'\}))\theta$ is said to be the resolvent of the clauses $C$ and $D$ whenever $C$ and $D$ are standardised apart, $A \in C$, $\neg A' \in D$, $\theta$ is the mgu of $A$ and $A'$. That is to say that $\langle A\theta, \neg A'\theta \rangle$ is a complementary pair. The resolvent of clauses $C$ and $D$ is denoted $(C \cdot D)$ when the complementary pair of literals is unspecified. The '$\cdot$' operator is commutative, non-associative and non-distributive.

Let $T$ be a clausal theory. Robinson [34] defined the function $\mathcal{R}^n(T)$ recursively as follows. $\mathcal{R}^0(T) = T$. $\mathcal{R}^n(T)$ is the set of all resolvents constructed from pairs of clauses in $\mathcal{R}^{n-1}(T)$. Robinson showed that $T$ is unsatisfiable if and only if there is some $n$ for which $\mathcal{R}^n(T)$ contains the empty clause ($\square$).

# References

[1] R.B. Banerji. Learning in the limit in a growing language. In *IJCAI-87*, pages 280–282, Los Angeles, CA, 1987. Kaufmann.

[2] C. Bennett. Logical depth and physical complexity. In R. Herken, editor, *The Universal Turing Machine A Half Century Survey*, pages 227–257. Kammerer and Unverzagt, Hamburg, 1988.

[3] I. Bratko. *Prolog for artificial intelligence.* Addison-Wesley, London, 1986.

[4] W. Buntine. Generalised subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.

[5] R. Carnap. *The Continuum of Inductive Methods.* Chicago University, Chicago, 1952.

[6] G. Chaitin. *Information, Randomness and Incompleteness - Papers on Algorithmic Information Theory.* World Scientific Press, Singapore, 1987.

[7] Dietterich. Limitations of inductive learning. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 124–128, San Mateo, CA, 1989. Morgan-Kaufmann.

[8] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. In *COLT 88: Proceedings of the Conference on Learning*, pages 110–120, Los Altos, CA, 1988. Morgan-Kaufmann.

[9] S. Feferman et al., editor. *Gödel's Collected Works.* Oxford University Press, Oxford, 1980.

[10] M.S. Fox and J. McDermott. The role of databases in knowledge-based systems. Technical Report CMU-RI-TR-86-3, Carnegie-Mellon University, Robotics Institute, Pittsburgh, PA, 1986.

[11] A.M. Frisch and C.D. Page. On inductive generalisation with taxonomic background knowledge. Technical report, University of Illinois, Urbana, 1989.

[12] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter System I. *Monats. Math. Phys.*, 32:173–198, 1931.

[13] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[14] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial intelligence*, 36:177 – 221, 1988.

[15] J. Hayes-Michie. News from Brainware. *Pragmatica*, 1:10–11, 1990.

[16] H. Ishizaka. Learning simple deterministic languages. In *Computational learning theory: proceedings of the second annual workshop*, San Mateo, CA, 1989. Kaufmann.

[17] R.A. Kowalski. *Logic for Problem Solving.* North Holland, 1980.

[18] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[19] X. Ling and M. Dawes. Theory reduction with uncertainty: A reason for theoretical terms. Technical Report 271, University of Western Ontario, 1990.

[20] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1984.

[21] M.J. Maher. Equivalences of logic programs. In *Proceedings of Third International Conference on Logic Programming*, Berlin, 1986. Springer.

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

[23] H. Mortimer. *The Logic of Induction.* Ellis Horwood, Chichester, England, 1988.

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

[25] S.H. Muggleton. Inverting the resolution principle. In *Machine Intellience 12 (in press).* Oxford University Press, 1988.

[26] S.H. Muggleton. A strategy for constructing new predicates in first order logic. In *Proceedings of the Third European Working Session on Learning*, pages 123–130. Pitman, 1988.

[27] S.H. Muggleton. *Inductive Acquisition of Expert Knowledge.* Addision-Wesley, Wokingham, England, 1990.

[28] S.H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

[29] S.H. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha.

[30] T. Niblett. A study of generalisation in logic programs. In *EWSL-88*, London, 1988. Pitman.

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

[32] J.R. Quinlan. Discovering rules from large collections of examples: a case study. In D. Michie, editor, *Expert Systems in the Micro-electronic Age*, pages 168–201. Edinburgh University Press, Edinburgh, 1979.

[33] J.R. Quinlan. Learning relations: comparison of a symbolic and a connectionist approach. Technical Report 346, University of Sydney, 1989.

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

[35] C. Rouveirol and J-F Puget. A simple and general solution for inverting resolution. In *EWSL-89*, pages 201–210, London, 1989. Pitman.

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

[37] E.Y. Shapiro. *Algorithmic program debugging.* MIT Press, 1983.

[38] E.H. Shortliffe and B. Buchanan. A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379, 1975.

[39] S. Slocombe, K. Moore, and M. Zelonf. Engineering expert systems applications. In *Proceedings of the Annual Conference of the BCS Specialist Group on Expert Systems.* British Computer Society, London, 1986.

[40] R.J. Solomonoff. A formal theory of inductive inference. *J. Comput. Sys.*, 7:376–388, 1964.

[41] L. Sterling and E. Shapiro. *The art of Prolog: advanced programming techniques.* MIT-Press, Cambridge, MA, 1986.

[42] A. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, pages 161–228, 1939.

[43] A. Turing. The automatic computing engine. *Lecture to the London Mathematical Society*, 1947.

[44] R. Wirth. Completing logic programs by inverse resolution. In *EWSL-89*, pages 239–250, London, 1989. Pitman.