Inverse entailment and Progol

Stephen MUGGLETON
Oxford University Computing Laboratory,
Wolfson Building,
Parks Road,
Oxford, OX1 3QD,
United Kingdom.

Abstract This paper firstly provides a re-appraisal of the development of techniques for inverting deduction, secondly introduces Mode-Directed Inverse Entailment (MDIE) as a generalisation and enhancement of previous approaches and thirdly describes an implementation of MDIE in the Progol system. Progol is implemented in C and available by anonymous ftp. The re-assessment of previous techniques in terms of inverse entailment leads to new results for learning from positive data and inverting implication between pairs of clauses.

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

1 Introduction

Since its inception in this journal [31] Inductive Logic Programming (ILP) has grown to become a substantial sub-area of both Machine Learning and Logic Programming (see [43]). The success of the subject lies partly in the choice of the core representation language of logic programs. Least Herbrand models of logic programs [26] fit neatly with the distinction between examples and conjectured theories in inductive inference. The syntax of logic programs provides modular blocks which, when added or removed, generalise or specialise the program. Depth-bounded Prolog interpreters, used for theorem-proving, allow efficient testing of hypothesised Horn clause theories. Most importantly, Turing-equivalence of logic programs is allowing a broader range of Machine Learning applications in ILP than was possible with more restrictive representations.

Recent research in ILP has spawned a variety of new theoretical topics. These include the problem of inverting resolution [37, 61, 54], inversion of clausal implication [22, 14, 40], predicate invention [36], closed-world specialisation [1] and U-learnability [42]. As with any subject, the diversity of sub-topics can be better understood by following the development of a particular line of ideas. The aims of this paper are firstly to provide a re-appraisal of the development of techniques for inverting deduction, secondly to introduce Mode-Directed Inverse Entailment (MDIE) as a generalisation and enhancement of previous approaches and thirdly to describe an implementation of MDIE in the Progol[1] system.

At each stage in the development of ILP there has been an attempt to solve existing technical restrictions of implemented systems. The five main approaches described in this paper are as follows.

1. Inverse resolution (IR) in propositional logic,
2. IR in first-order definite clause logic,
3. determinate relative least general generalisation,
4. inverse implication and
5. mode-directed inverse entailment.

The paper is structured as follows. First the logical and statistical setting for ILP are introduced (Section 2). This is followed by a synopsis of the results and restrictions for approaches 1 to 4 (Sections 3 to 6). The remainder of the paper (Sections 7 to 12) deals with theoretical and practical aspects of mode-directed inverse entailment. Instructions for obtaining Progol by anonymous ftp are given in Section 11. The paper closes with a discussion of research issues related to inverse entailment. Standard definitions taken from Logic Programming and

[1] Prolog inverted in the middle.

ILP are given in Appendix A. In Appendix B a statistical setting for ILP is described. Properties of the subsumption lattice are described in Appendix C. The algorithms used in Progol are given in Appendix D. A table of Progol's runtimes on various data sets is presented in Appendix E.

2 Logical and statistical setting for ILP

Deductive inference derives consequences E from a prior theory T. Thus if T says that all swans are white, E might state that a particular swan is white. Inductive inference derives a general belief T from specific beliefs E. After observing one or more white swans T might be the conjecture that all swans are white. In both deduction and induction T and E must be consistent and

$$T \models E. \tag{1}$$

The requirement of consistency means that the observation of a black swan rules out conjecture T. Inductive inference is, in a sense, the inverse of deduction. However, deductive inference proceeds by application of sound rules of inference, while inductive inference typically involves unsound conjecture. Such conjectures have at best statistical support from observed data. However, the association of probability values with hypotheses requires the assumption of a prior probability distribution over the hypothesis language. Occam's razor can be taken as an instance of a distribution which assigns higher prior probability to simpler hypotheses. It has been shown [4] that without such distributional assumptions the class of all logic programs is not even PAC-predictable. On the other hand, it has recently been demonstrated [42] that the class of all time-bounded logic programs is polynomial-time learnable (U-learnable) under fairly broad families of prior probability distributions. Appendix B gives more details of the relationship between data, posterior probabilities and U-learnability.

Within ILP it is usual to separate the elements of (1) into examples (E), background knowledge (B), and hypothesis (H). These have the relationship

$$B \wedge H \models E. \tag{2}$$

B, H and E are each logic programs. E usually consists of ground unit clauses of a single target predicate. E can be separated into E^+, ground unit definite clauses and E^-, ground unit headless Horn clauses. However, the separation into B, H and E is a matter of convenience, as the following example shows.

Example 1 White swans. *The swan example might be represented using the following logic program.*

$$E^+ = \left\{ \begin{array}{l} white(swan1) \leftarrow \\ swan(swan1) \leftarrow \end{array} \right.$$

$$E^- = \left\{ \begin{array}{l} black(swan2) \leftarrow \\ swan(swan2) \leftarrow \end{array} \right.$$

$$B = \left\{ \leftarrow black(X), white(X) \right.$$

$$H = \left\{ white(X) \leftarrow swan(X) \right.$$

Relationship (2) does not hold since $swan(swan1)$ is not entailed by $B \wedge H$. It does not help to argue that $swan(swan1)$ is background knowledge, since this is an observations about swan1. E^- does not contain headless Horn clauses, although together with B it refutes H. These problems can most simply be avoided by dropping all but the restriction that B, H and E are arbitrary logic programs.

3 Inverse resolution in propositional logic

The idea of carrying out induction by inverting deduction was first investigated in depth mathematically by the 19th century political economist and philosopher of science Stanley Jevons [16][2]. Jevons solved by tabulation the "Inverse or Inductive Problem" involving two propositional symbols. The following quote from Jevons' book on inductive inference [16] is both modern-sounding and relevant to the problems addressed in this paper.

> Induction is, in fact, the inverse operation of deduction, and cannot be conceived to exist without the corresponding operation, so that the question of relative importance cannot arise. Who thinks of asking whether addition or subtraction is the more important process in arithmetic? But at the same time much difference in difficulty may exist between a direct and inverse operation; the integral calculus, for instance, is infinitely more difficult than the differential calculus of which it is the inverse. Similarly, it must be allowed that inductive investigations are of a far higher degree of difficulty and complexity than any questions of deduction; ...

At the time of Jevons logicians, not yet persuaded of Boole's algebraic approach to logic, employed an array of inference rules derived from Aristotelian syllogisms. Robinson [53] was later to show that deductive inference in first-order predicate calculus could be effected by a single rule of inference, that of resolution. Inductive inference based on inverting resolution in propositional logic was first discussed in [32] (originally a technical report from 1987) as an analysis of the inductive inference rules within the Duce system [29].

[2]George Boole's algebraic approach to deduction inspired Jevons to use truth-functional tabulations to design and build a logical calculator [15]. Jevons' mechanical *Organon* is complete for deciding satisfiability of propositional clauses in 4 variables, and can be found in the Museum of Scientific Instruments in Oxford.

3.1 Inductive inference rules

Duce had six inductive inference rules. Four of these were concerned with definite clause propositional logic. In the following description of the inference rules lower-case letters represent propositional variables and upper-case letters represent conjunctions of propositional variables.

$$\textbf{Absorption:}\ \frac{p \leftarrow A, B \qquad q \leftarrow A}{p \leftarrow q, B \qquad q \leftarrow A}$$

$$\textbf{Identification:}\ \frac{p \leftarrow A, B \qquad p \leftarrow A, q}{q \leftarrow B \qquad p \leftarrow A, q}$$

$$\textbf{Intra-construction:}\ \frac{p \leftarrow A, B \qquad\qquad\qquad p \leftarrow A, C}{q \leftarrow B \qquad p \leftarrow A, q \qquad q \leftarrow C}$$

$$\textbf{Inter-construction:}\ \frac{p \leftarrow A, B \qquad\qquad\qquad q \leftarrow A, C}{p \leftarrow r, B \qquad r \leftarrow A \qquad q \leftarrow r, C}$$

Each of Duce's rules is superficially similar to that of a deductive rule of inference of the form

$$\frac{X}{Y}$$

Such a deductive inference rule would be called sound if and only if X entailed Y. We will call a rule of inference *inductively sound* if and only if Y logically entails X, or equivalently $\overline{X}$ entails $\overline{Y}$. A set of inductive inference rules will be written with an overline as $\overline{I}$. Each clause above the line is either a resolvent of two clauses below the line or is itself found below the line. Duce's inference rules invert single-depth applications of resolution. Using the rules a set of resolution-based trees for deriving the examples can be constructed backwards from their roots. The set of leaves of the trees represent a theory from which the examples can be derived. In the process new proposition symbols, not found in the examples, can be "invented" by the intra- and inter-construction rules.

3.2 Completeness

Continuing the analogy with deduction we might write

$$\overline{X} \vdash_{\overline{I}} \overline{Y}$$

to say that theory Y is derivable using inductive inference rules $\overline{I}$ from examples X. There are two senses in which a set of inference rules $\overline{I}$ may be said to be complete.

Definition 2 Weak completeness. *Let the example language $\mathcal{E}$ and hypothesis language $\mathcal{H}$ both be subsets of the first-order predicate calculus and let $\overline{I}$ be a set of inductive inference rules. $\overline{I}$ is said to be weak complete for $\mathcal{E}$ and $\mathcal{H}$ if and only if for each $H \subseteq \mathcal{H}$ there exists $E \subseteq \mathcal{E}$ such that $\overline{E} \vdash_{\overline{I}} \overline{H}$.*

In [32] it was shown that $\overline{I}$ consisting of only *absorption* and *intra-construction* is weak complete under particular hypothesis and example language restrictions.

Definition 3 Strong completeness. *Let the example language $\mathcal{E}$ and hypothesis language $\mathcal{H}$ both be subsets of the first-order predicate calculus and let $\overline{I}$ be a set of inductive inference rules. $\overline{I}$ is said to be strong complete for $\mathcal{E}$ and $\mathcal{H}$ if and only if for each $H \subseteq \mathcal{H}$ and $E \subseteq \mathcal{E}$ $H \models E$ implies $\overline{E} \vdash_{\overline{I}} \overline{H}$.*

The four Duce inference rules in Section 3.1 are not strong complete for definite clause propositional calculus.

3.3 Occam compression

In Duce every application of an inductive inference rule $\frac{X}{Y}$ was chosen to maximise information compression.

Definition 4 Occam compression. *Let X, Y be wffs for which $Y \models X$ and $X \wedge Y \not\models \Box$. Let $|X|$ and $|Y|$ be the number of bits required to encode X and Y. The Occam compression of X relative to Y is $|X| - |Y|$.*

Suppose $|P| = b.symbols(P)$ where $symbols(P)$ is the number of propositional symbol occurrences in P and b is the number of bits to encode each such occurrence. With reference to Appendix B, an encoding is the expression of a prior distribution. $F(P)$ expresses the relative frequency with which the teacher chooses P as target concept. Assume the learner knows $F(P)$ and uses it as a prior distribution on $\mathcal{H}$. Then according to Shannon and Weaver [56] $|P|$ is $-log_2 F(P)$ and

$$F(P) = 2^{-|P|}$$

Note that since this is an exponential-decay distribution, in the situation in which the learner knows $F(P)$, the results in [42] show that the class of all time-bounded logic programs are polynomial-time learnable (U-learnable). However, note also that if the teacher's prior is known to the learner then on average theories chosen by the teacher have extremely low information content. Alternatively this might be viewed as the expectation that only a small augmentation of an existing theory is expected from any short presentation of the teacher's examples.

Remark 5 *Let E be a wff and $\mathcal{H}$ be a set of wffs containing E such that for each $H \in \mathcal{H}$ it is the case that $H \models E$ and $H \wedge E \not\models \Box$. Let H_{max} have maximum compression within $\mathcal{H}$ relative to E and let H_0 have compression 0 relative to E.*

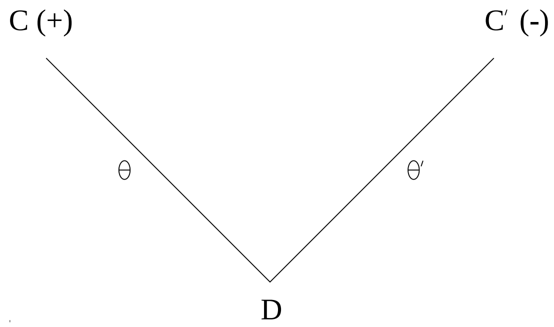

Figure 1: Single resolution.

H_{max} has maximum posterior probability and H_0 has posterior probability equal to E.
Proof. *According to Equation (6) in Appendix B.2*

$$\frac{p(H|E)}{p(E|E)} = \frac{p(H)}{p(E)} = 2^{|E|-|H|}.$$

$p(H|E)$ is maximal when $|E| - |H|$ is maximal. When $|E| - |H| = 0$ then $p(H|E) = p(E|E)$. □

The hypothesis with maximum posterior probability (H_{max}) has maximum expected predictive accuracy.

4 Inverse resolution in first-order logic

Inverse resolution was lifted to first-order predicate calculus in [37]. This involved algebraic inversion of the equations of resolution below.

$$\begin{aligned} D &= (C \cup C')\theta\theta' \\ l\theta &= \overline{l'\theta'} \end{aligned}$$

Figure 1 shows a resolution step. D is derived at the base of the 'V' given the clauses on the arms. In contrast, a 'V' inductive inference step 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 1 the literal resolved on is positive (+) in C and negative (-) in C'. Duce's absorption rule constructs C' from C and D, while the identification rule derives C from C' and D.

Since algebraic inversion of resolution has a complex non-deterministic solution only a restricted form of absorption was implemented in Cigol[3]. However,

[3]logiC backwards.

it was shown independently in [31] and [54] that there is a unique most-specific solution for 'V' inductive inference rules. That is

$$C' \downarrow \ = (D \cup l\theta)$$

where θ is such that $C\theta \subseteq D$. Rather than inverting the equations of resolution we might consider resolution from the model-theoretic point of view. That is

$$C \wedge C' \models D. \tag{3}$$

Applying the deduction theorem gives a deductive solution for absorption.

$$C \wedge \overline{D} \models \overline{C'}$$

This is a special case of *inverting implication* (Section 7). Since D and C' are clauses, $\overline{D}$ and $\overline{C'}$ are conjunctions of ground skolemised literals. The most specific solution for C' corresponds to the most general solution for $\overline{C'}$, i.e. when $\overline{C'}$ contains the maximum set of literals derivable from $C \wedge \overline{D}$. However, this solution is neither restricted to single-depth resolutions, nor is the clause cardinality finitely bounded.

Example 6 Recursive list membership. *Let* $C = member(X, [X|Y])$ *and* $D = member(2, [1, 2, 3])$.

$$\begin{aligned} C \wedge \overline{D} &\models \overline{member(2, [1, 2, 3])} \\ &\models member(1, [1, 2, 3]) \\ &\models member(2, [2, 3]) \\ &\models member(3, [3]) \\ &\models \ldots \end{aligned}$$

Though the clause $C' = member(2, [1, 2, 3]) \leftarrow member(1, [1, 2, 3]), ..$ *maintains Relationship (3), there are at least 3 derivation steps to D.* C' *is θ-subsumed by all single-step resolution solutions.* $\overline{C'}$ *also contains the infinite sequence of atoms* $member(3, [3, 3])$, $member(3, [3, 3, 3]), \ldots$.

Owing to the weak completeness results for the Duce inductive inference rules (Section 3.2) only absorption and intra-construction were implemented in Cigol.

4.1 Compression

Like Duce, Cigol used Occam compression (Definition 4) to guide the choice of inverse resolution steps. The encoding measure was the total number of predicate and function symbol occurrences in a logic program. Like Duce, each such inverse resolution step was only allowed if it produced a positive compression value. This lead to two difficulties.

1. **Local generalisation.** Consider the recursive multiplication clause

$$mult(A, B, C) \leftarrow dec(A, D), mult(D, B, E), plus(E, B, C).$$

When given a large set of ground instances of valid multiplications, compression is only achievable after a series of inverse resolution steps, in which all steps except the last do not produce compression.

2. **Learning from positive examples.** In [30] it was noted that the compression measure used in Cigol did not allow learning from only positive data since the simplest possible hypothesis, say $\forall X.p(X)$, will always be consistent. Alternative compression measures were suggested in [30, 44, 5, 9]. These measures are closely allied to Rissanen's Minimal Description Length (MDL) Principle [52, 24].

The first problem was addressed by considering the inversion of multiple resolution steps by *saturating* clauses [55, 54, 31, 13]. Clause saturation is closely related to the techniques of inverse entailment described in Section 7. However, since saturation is based on inverting resolution proof steps, it cannot deal with built-in predicates. Nevertheless, the interpretations of such predicates can be computed by calling C functions. The Progol system (Sections 8 to 11) uses mode declarations to access such interpretations.

4.2 Learning from positive data

The second problem is of a different nature. When learning from only positive data, predictive accuracy will be maximised by choosing the most general consistent hypothesis since this will always agree with new data. However, in applications such as grammar learning [25, 50], only positive data are available, though the grammar which produces all strings is not an acceptable hypothesis. Let us then suppose a modification to the U-learning setting given in Appendix B. The teacher still draws instances randomly from distribution G but only gives them to the learner if they are positive examples of the target T. In this setting we would need to find a tradeoff between the generality and complexity of an hypothesis. First let us define a measure of the generality of an hypothesis.

Definition 7 Generality measure. *Let H be a wff and G be a probability distribution over a (possibly infinite) set of wffs X. The generality g of H is defined as*

$$g(H) = \sum_{x \in X, H \models x} G(x).$$

Since G is a probability distribution it follows for every $H \in \mathcal{H}$ that $0 \leq g(H) \leq 1$. $g(H)$ is the probability that an instance drawn randomly from G will be entailed by H. Note therefore that $g(\Box) = 1$, $g(\blacksquare) = 0$ and $T_1 \models T_2$ implies $g(T_1) \geq g(T_2)$.

Clearly for infinite instance spaces $g(H)$ cannot be calculated exactly. However, according to the Central Limit Theorem, given a sufficiently large random sample S from G, the proportion of S entailed by H is an arbitrarily good estimate of $g(H)$. Now consider the following probability distribution.

$$f_m(H) = c.2^{-|H|}(1 - g(H))^m.$$

m is the number of examples so far and c is a normalising constant to ensure that for $H \in \mathcal{H}$ the function f_m sums to 1. f_m trades off the complexity of an hypothesis against its generality. Note that since f_m varies with m, it cannot be viewed as a prior distribution over hypotheses. As with MDL f_m increases the discrimination against over-generality with increasing numbers of examples. When used to choose between hypotheses given positive-only data f_m has the following convergence property.

Theorem 8 Finite elimination of false conjectures with positive-only data. *Let T be an element of the set of wffs $\mathcal{H}$ and let G be a probability distribution over the set of wffs X such that $x \in X$ has non-zero probability in G if and only if $T \models x$. Let T' be the minimal complexity expression of T in $\mathcal{H}$. Let $\langle x_1, x_2, \ldots \rangle$ be an infinite series of wffs drawn randomly according to G. Let $f_i(H)$ have value $2^{-|H|}(1 - g(H))^i$ for all those H in $\mathcal{H}$ which entail each x_j, $1 \leq j \leq i$, and have value 0 otherwise. Let H be any element of $\mathcal{H}$ such that H does not entail the same subset of X as T. Then there exists a finite natural number k such that $f_k(H) < f_k(T')$.*

Proof. *Suppose there is an H for which there is no such k. It cannot be the case for H that $g(H) > g(T')$ and $|H| > |T'|$ since otherwise for all i, $i \geq 0$, $f_i(H) < f_i(T')$. Therefore suppose $g(H) > g(T')$ and $|H| \leq |T'|$. But then since $(1 - g(H))^i$ decreases monotonically with i there must exists k such that for all $j \geq k$ it is the case that $f_j(H) < f_j(T')$. Therefore it must be that $|H| > |T'|$ and $g(H) < g(T')$. But then there exists k and x_k such that $T' \models x_k$ and $H \not\models x_k$ and therefore $f_k(H) = 0 < f_k(T')$. This contradicts the assumption and completes the proof.* $\Box$[4]

f_m provides the basis for a simplified version of the compression models defined in [30, 44].

Definition 9 Positive-only compression. *Let H be a wff and G be a distribution over instance space X. Let $E \subseteq X$ be a set of m examples of H. Let $|H|$ and $|E|$ be the number of bits required to encode H and E. The positive-only*

[4] At first sight, this theorem appears to clash with the fundamental result of Gold [10] that not even the regular languages can be identified in the limit from positive data alone. However, it cannot be guaranteed after any finite number of examples that all H which are not over-general have lower values of f_m than T'.

compression of E to H is

$$\begin{aligned} pcomp(H,E) &= log_2 \frac{f_m(H)}{f_m(E)} \\ &= |E| - |H| - m(log_2(1-g(E)) - log_2(1-g(H))) \\ &\approx |E| - |H| + m log_2(1-g(H)). \end{aligned}$$

The approximation in the last line applies for small m, in which case $g(E)$ is close to 0.

5 Relative least general generalisations

One commonly advocated approach to learning from positive data is that of taking relative least general generalisations (rlggs) of clauses (see Appendix C). Suppose, as in the last section, that the teacher chooses target T and presents to the learner examples $E = \{x_1, x_2, .., x_m\}$. Given background knowledge B, $H = rlgg_B(E)$ will be the hypothesis within the relative subsumption lattice with the fewest possible errors of commission (instances $x \in X$ for which $H \models x$ and $T \not\models x$). This approach to learning from positive data has the following problems.

1. **Arbitrary background knowledge.** Plotkin [47] showed that with unrestricted definite clause background knowledge B there may not be any finite $rlgg_B(E)$.
2. **Extensional background knowledge.** Suppose B and E consist of n and m ground unit clauses respectively. In the worst case the number of literals in $rlgg_B(E)$ will be $(n+1)^m$, making the construction intractable for large m.
3. **Multiple clause hypothesis.** Target concepts with multiple clauses cannot be learned since $rlgg_B(E)$ is a single clause.

In contrast, none of these problems occur if H is chosen from the set of all definite clause theories $\mathcal{H}$ using maximum positive-only compression (Definition 9). Suppose $E \in \mathcal{H}$ and H is the hypothesis with maximum positive-only compression. As with $rlgg_B(E)$, H will be maximally specific among clauses of the same complexity. Also H will always have complexity of at most that of E. Lastly H can be a multiple clause hypothesis.

5.1 Golem

Golem was designed to overcome the search problems of Cigol (Section 4.1). The unique construction of rlggs contrasts with the highly non-deterministic choices involved in inverting a resolution step.

Golem used extensional background knowledge to avoid the problem of non-finite rlggs. Extensional background knowledge B can be generated from intensional background knowledge B' by generating all ground unit clauses derivable from B' in at most h resolution steps. The parameter h is provided by the user. The rlggs constructed by Golem were forced to have only a tractable number of literals by requiring that $\mathcal{H}$ contain definite clause theories that were ij-determinate. The idea behind ij-determinacy is as follows. Let C be a definite clause of the form

$$\forall \vec{X}.h \leftarrow b_1, b_2, .., b_n$$

where $\vec{X}$ is the vector of all variables within C. Suppose that $\vec{Y}$ are the variables in the head of C and $\vec{Z}$ are the variables found only in the body of C. C can equivalently be written

$$\forall \vec{Y}.h \leftarrow (\exists \vec{Z} b_1, b_2, .., b_n).$$

Determinacy is a constraint which restricts the quantification on variables $\vec{Z}$ in the body of definite clauses to Hilbert ϵ^* (exists exactly one) quantification. This is equivalent to requiring that predicates in the background knowledge must represent functions. Thus for every example e and hypothesised clause C there must exist at most one valid substitution for the variables $\vec{Z}$ in the body of C. j-determinate clauses are constrained to having at most j variables in any literal. ij-determinate clauses are further restricted that each variable has depth at most depth i. For variable v the depth $d(v)$ is defined recursively as follows.

Definition 10 Depth of variables.

$$d(v) = \begin{cases} 0 & \textit{if } v \textit{ is in the head of } C \\ (max_{u \in U_v} d(u)) + 1 & \textit{otherwise} \end{cases}$$

where U_v are the variables in atoms in the body of C containing v.

Multiple clause theories could be learned by Golem due to the use of negative examples. Each clause was built from the rlgg of a set of positive examples. Negative examples were used to stop rlggs becoming over-general.

5.2 Application experience

Golem was the first ILP system to be applied to a wide variety of real-world applications. These included the construction of a satellite fault diagnosis model [8], the design of a qualitative physics model [2], finite-element mesh design [6], protein secondary structure prediction [39] and structure-activity prediction for drugs [18]. In the qualitative physics domain Golem was hampered in requiring a large tabulation of the QSIM simulator. The determinacy restriction was inappropriate in the finite element mesh design application. The restrictions of Golem and other ILP algorithms are discussed in [35].

Golem was also applied to various list and number-theoretic learning tasks involving the construction of recursive theories. Learning recursive theories was awkward using Golem partly because intensional hypothesised base cases could not be used to augment the entirely extensional background knowledge. Also Golem's search was through the subsumption lattice, rather than the lattice of implication between clauses.

6 Implication between clauses

In [47] Plotkin noted that if clause C θ-subsumes clause D (or $C \preceq D$) then $C \rightarrow D$. However, he also notes that $C \rightarrow D$ does not imply $C \preceq D$, as shown by the following example.

Example 11 Implication and subsumption. *Consider the following clauses.*

$$\begin{aligned} C = &\quad nat(s(X)) \leftarrow nat(X) \\ D = &\quad nat(s(s(Y))) \leftarrow nat(Y) \end{aligned}$$

$C \rightarrow D$ but not $C \preceq D$.

Although efficient methods are known [20] for enumerating every clause C which θ-subsumes an arbitrary clause D, this is not the case for clauses C which imply D. This is known as the problem of inverting implication between clauses. The inability to invert implication between clauses limits the completeness of inverse resolution and rlggs since θ-subsumption is used in place of clause implication in both.

Gottlob [11] proves a number of properties concerning implication between clauses. The following lemma is notable.

Lemma 12 Gottlob's lemma. *Let C, D be two non-tautological clauses. Let C^+, C^- be the sets of positive and negative literals of clauses C and D^+, D^- be the same for D. $C \rightarrow D$ implies that $C^+ \preceq D^+$ and $C^- \preceq D^-$.*

In an attempt to solve the inverting implication problem Lapointe and Matwin [22] introduced sub-unification, a process of matching sub-terms in D to produce C. They demonstrate that sub-unification is able to construct recursive clauses from fewer examples than would be required by ILP systems such as Golem [38] and FOIL [49]. Although the operations described by Lapointe and Matwin are shown to work on a number of examples it is not clear how general the mechanism is. Various general properties of implication between clauses are investigated in [33]. In particular it is shown that Lee's subsumption lemma [23] has the following corollary.

Corollary 13 Implication and recursion. *Let C, D be clauses. $C \rightarrow D$ if and only if either D is a tautology or $C \preceq D$ or there is a clause E such that $E \preceq D$ where E is constructed by repeatedly self-resolving C.*

Thus the difference between θ-subsumption and implication between C and D is only pertinent when, as in Example 11, C can self-resolve. Attempts were made to a) extend inverse resolution [33] and b) use a mixture of inverse resolution and lgg [14] to solve the problem. The extended inverse resolution method in [33] suffers from the same problems of non-determinacy as Cigol. Idestam-Almquist's [14] use of lgg suffers from the standard problem of intractably large clauses (see Section 5). Both approaches are incomplete for inverting implication, though Idestam-Almquist's technique is complete for a restricted form of entailment called T-implication.

In [40] it is shown that for certain recursive clauses D all the clauses C which imply D also θ-subsume a logically equivalent clause D'. Up to renaming of variables every clause D has at most one most specific form of D' in the θ-subsumption lattice. D' is called the self-saturation of D. The self-saturation of D in Example 11 is simply $C \cup D$. However, it is shown in [40] that there exist definite clauses which have no finite self-saturation.

6.1 Inverting entailment between clauses

This section gives a complete and efficient method for inverting implication between function-free definite clauses. The techniques used are based on inverting entailment using the deduction theorem. First we define definite sub-saturants.

Definition 14 Definite sub-saturants. *Let $D = h \leftarrow b_1, .., b_n$ be a definite clause. Let $\mathcal{B}(\overline{D})$ be the Herbrand base of $\overline{D}$ restricted to the predicate symbol of h and let $\mathcal{M}(\overline{D})$ be the minimal Herbrand model of $\overline{D}$. Let $desk(a)$ be the atom a with skolem constants in $\overline{D}$ replaced by their corresponding variables in D. Let $\mathcal{A}(D)$ be $\mathcal{B}(\overline{D}) - \mathcal{M}(\overline{D})$. The sub-saturants of D, $\mathcal{S}(D)$ are the set of all definite clauses $desk(a) \leftarrow b_1, .., b_n$ for which $a \in \mathcal{A}(D)$.*

Although arbitrary definite clauses can have an infinite sub-saturant set, this is not so for function-free definite clauses. It is now shown for function-free clauses that if k is a bound on the arity of predicates then the cardinality of the sub-saturant set is polynomially bounded in the number of variables in D.

Remark 15 Cardinality of sub-saturant set. *Let D be a function-free definite clause, k be the arity of the predicate symbol in the head of D, n be the number of variables in D and $\mathcal{S}(D)$ be the sub-saturants of D. The cardinality of $\mathcal{S}(D)$ is at most n^k.*
Proof. *The arguments of the heads of clauses in $\mathcal{S}(D)$ are k-length permutations of variables in D. There are n^k such permutations.* $\square$

We now present the main theorem concerning sub-saturants.

Theorem 16 *Let C and D be definite non-tautological clauses and $\mathcal{S}(D)$ be the sub-saturants of D. $C \models D$ only if there exists C' in $\mathcal{S}(D)$ such that $C \preceq C'$.*

Proof. *Suppose $C \models D$ and there does not exist C' in $\mathcal{S}(D)$ such that $C \preceq C'$. According to Lemma 12 the heads of C and D have the same predicate symbol. Since $C \models D$ it follows that $C \wedge \overline{D}$ is not satisfiable. According to Herbrand's theorem this is the case if and only if $C \wedge \overline{D}$ has no Herbrand model. According to Lemma 12 the body of C θ-subsumes the body of D and therefore there exists a ground (skolemised) substitution θ for which all elements in the body of C are true in the least model of $\overline{D}$. Therefore with substitution θ the head of C must be false in the least Herbrand model of $\overline{D}$ since otherwise $C \wedge \overline{D}$ has a Herbrand model. But according to the construction in Definition 14 for every such C with the same predicate symbol as D there is a C' in $\mathcal{S}(D)$ such that $C \preceq C'$. This contradicts the assumption and completes the proof.* $\square$

This theorem can be used to efficiently enumerate all function-free definite clauses C such that $C \models D$. First the finite set of sub-saturants $\mathcal{S}(D)$ is constructed. Then the clauses which θ-subsume any clause in $\mathcal{S}(D)$ are enumerated using an efficient interleaved enumeration of the subsumption lattice. Since function-free first-order predicate calculus is decidable the clauses C for which $C \models D$ can be enumerated by testing $C \wedge D \vdash \square$.

Example 17 Factorial. *$x! = (x-2)!(x-1)x$ is an overly specific recurrence formula for the factorial function. This formula can be represented by the clause*

$$D = f(I,J) \leftarrow d(I,K), d(K,L), f(L,M), m(K,M,N), m(I,N,J)$$

where the predicate symbols are f =factorial, d =decrement, m =multiply. Since there are 14 variables in D it follows from Remark (15) that the cardinality of $\mathcal{S}(D)$ is at most $14^2 = 196$. $\mathcal{S}(D)$ contains the clause

$$C' = f(K,N) \leftarrow d(I,K), d(K,L), f(L,M), m(K,M,N), m(I,N,J).$$

The following clause C which implies D (but does not θ-subsume D) corresponds to the most general recurrence for factorial, $x! = (x-1)!x$.

$$C = f(K,N) \leftarrow d(K,L), f(L,M), m(K,M,N).$$

The following example demonstrates how clauses with function symbols, such as those in Example 11, can be dealt with as though they were function-free by using *flattening* [54].

Example 18 Flattening and inverse implication. *The clause $D = nat(s(s(X))) \leftarrow nat(X)$ can be flattened to the function-free clause $D' = nat(V) \leftarrow s(V,W), s(W,X), nat(X)$ where s is defined as $s(X, s(X))$. There are 2 sub-saturants of D', which are D' itself and $C'' = nat(W) \leftarrow s(V,W), s(W,X), nat(X)$ which is θ-subsumed by $C' = nat(W) \leftarrow s(W,X), nat(X)$. C' can be unflattened to the following clause which implies but does not θ-subsume D.*

$$C = nat(s(X)) \leftarrow nat(X)$$

7 Inverting entailment

Inverse resolution and other subsumption oriented approaches to induction have been re-assessed in previous sections of this paper. It has been demonstrated that a great deal of clarity and simplicity can be achieved by approaching the problem from the direction of model-theory rather than resolution proof-theory. In Duce an inductive inference rule $\frac{X}{Y}$ is sound in the deductive sense if viewed as stating the relationship $\overline{X} \models \overline{Y}$. In Cigol all solutions for absorption are found by simply rewriting the inductive specification $C \wedge C' \models D$ by the equivalent deduction oriented relationship $C \wedge \overline{D} \models \overline{C'}$. Lastly, it has been shown in this paper that a solution to Plotkin's 25 year old problem of generalising θ-subsumption can be achieved with relative ease by simply viewing solutions for C in $C \models D$ (given D) as clauses which eliminate Herbrand models of $C \wedge \overline{D}$.

Let us now consider the general problem specification of ILP (Section 2) in this light. That is, given background knowledge B and examples E find the simplest consistent hypothesis H (where simplicity is measured relative to a prior distribution) such that

$$B \wedge H \models E. \tag{4}$$

It was demonstrated in Example 1 that in general B, H and E could be arbitrary logic programs. Each clause in the simplest H should explain at least one example, since otherwise there is a simpler H' which will do. Consider then the case of H and E each being single Horn clauses. This can now be seen as a generalised form of absorption (Relation (3) in Section 4) and rearranged similarly to give

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

Since H and E are each single clauses, $\overline{H}$ and $\overline{E}$ will be logic programs consisting only of ground skolemised unit clauses. Let $\overline{\bot}$ be the (potentially infinite) conjunction of ground literals which are true in all models of $B \wedge \overline{E}$. Since $\overline{H}$ must be true in every model of $B \wedge \overline{E}$ it must contain a subset of the ground literals in $\overline{\bot}$. Therefore

$$B \wedge \overline{E} \models \overline{\bot} \models \overline{H}$$

and so for all H

$$H \models \bot.$$

A subset of the solutions for H can be found by considering the clauses which θ-subsume $\bot$. The complete set of candidates for H can be found by considering all clauses which θ-subsume sub-saturants of $\bot$ (Section 6.1).

Example 19 Various examples of $\bot$. *Figure 2 shows various B, E and $\bot$. In the first case, the clauses which θ-subsume $\bot$ include all those which could be reached using first-order absorption (Section 4). In the second case the definite clauses which θ-subsume $\bot$ are those which could be reached by a first-order version of Duce's identification operator (Section 3.1). This form of identification is*

B	**E**	$\perp$
anim(X)$\leftarrow$ pet(X). pet(X)$\leftarrow$ dog(X).	nice(X)$\leftarrow$ dog(X).	nice(X) $\leftarrow$ dog(X), pet(X), anim(X).
hasbeak(X)$\leftarrow$ bird(X). bird(X)$\leftarrow$ vulture(X).	hasbeak(tweety).	hasbeak(tweety); bird(tweety); vulture(tweety).
white(swan1).	$\leftarrow$ black(swan1).	$\leftarrow$ black(swan1), white(swan1).
sentence([],[]).	sentence([a,a,a],[]).	sentence([a,a,a],[]) $\leftarrow$ sentence([],[]).

Figure 2: The most-specific clause ($\perp$) for various versions of background knowledge (B) and example (E).

a general form of Kakas et al's abduction [17] and is of central interest in "theory revision" (alterations in theory revision range over all definitions within a hierarchical set of predicates which reference each other). The third case demonstrates that constraints (headless Horn clauses) can be learned from negative examples since the clause

$$\leftarrow black(X), white(X)$$

θ-subsumes $\perp$. In the fourth case one of the clauses which θ-subsumes a sub-saturant of the flattened $\perp$ (see Example 18) is the DCG grammar rule

$$sentence([a|X], Y) \leftarrow sentence(X, Y).$$

8 The definite mode language

In general $\perp$ can have infinite cardinality. Progol uses mode declarations to constrain the search for clauses which θ-subsume $\perp$ (see last Section).

Definition 20 Mode declaration. *A mode declaration has either the form modeh(n,atom) or modeb(n,atom) where n, the recall, is either an integer, $n > 1$, or '*' and atom is a ground atom. Terms in the atom are either normal or place-marker. A normal term is either a constant or a function symbol followed by a bracketed tuple of terms. A place-marker is either +type, -type or #type, where type is a constant. If m is a mode declaration then $a(m)$ denotes the atom of m with place-markers replaced by distinct variables. The sign of m is positive if m is a modeh and negative if m is a modeb.*

For instance the following are mode declarations.

modeh(1,plus(+int,+int,-int))	modeb(*,append(-list,+list,+list)
modeb(1,append(+list,[+any],-list))	modeb(4,(+int > #int))

The recall is used to bound the number of alternative solutions for instantiating the atom. For simplicity, we assume in the following that all the modes have the recall '*', meaning all solutions. The following defines when a clause is within Progol's definite mode language $\mathcal{L}$.

Definition 21 Definite mode language. *Let C be a definite clause with a defined total ordering over the literals and M be a set of mode declarations. $C = h \leftarrow b_1, .., b_n$ is in the definite mode language $\mathcal{L}(M)$ if and only if 1) h is the atom of a modeh declaration in M with every place-marker +type and -type replaced by variables and every place-marker #type replaced by a ground term and 2) every atom b_i in the body of C is the atom of a modeb declaration in M with every place-marker +type and -type replaced by variables and every place-marker #type replaced by a ground term and 3) every variable of +type in any atom b_i is either of +type in h or of -type in some atom b_j, $1 \leq j < i$.*

Like Golem, Progol constructs clauses of bounded depth (see Definition 10 in Section 5.1).

Definition 22 Depth-bounded mode language. *Let C be a definite clause with a defined total ordering over the literals and M be a set of mode declarations. C is in $\mathcal{L}_i(M)$ if and only if C is in $\mathcal{L}(M)$ and all variables in C have depth at most i according to Definition 10.*

Example 23 Factorial revisited. *Reconsider Example 17 with M being*

modeh(,f(+int,-int))* *modeb(*,d(+int,-int)*
modeb(,f(+int,-int))* *modeb(*,m(+int,-int))*

The clause

$$f(A, B) \leftarrow d(A, C), f(C, D), m(A, D, B)$$

is only in $\mathcal{L}_i(M)$ for $i \geq 2$.

8.1 Most-specific clauses in $\mathcal{L}_i(M)$

Progol searches a bounded sub-lattice for each example e relative to background knowledge B and mode declarations M. The sub-lattice has a most general element ($\top$) which is the empty clause, $\Box$, and a least general element $\bot_i$ which is the most specific element in $\mathcal{L}_i(M)$ such that

$$B \wedge \bot_i \wedge \overline{e} \vdash_h \Box$$

where $\vdash_h \Box$ denotes derivation of the empty clause in at at most h resolutions.

Definition 24 Most-specific clause $\bot_i$. *Let h, i be natural numbers B be a set of Horn clauses, $e = a \leftarrow b_1, .., b_n$ be a definite clause, M be a set of mode declarations containing exactly one modeh m such that $a(m) \preceq a$ and $\bot$ be the most-specific (potentially infinite) definite clause such that $B \wedge \bot \wedge \overline{e} \vdash_h \Box$. $\bot_i$ is the most-specific clause in $\mathcal{L}_i(M)$ such that $\bot_i \preceq \bot$.*

Progol constructs $\bot_i$ using Algorithm 40 in Appendix D.1.

Theorem 25 Correctness of Algorithm 40. *Let h, i, B, M be defined as in Definition 24. Given h, i, B, e and M Algorithm 40 returns an alphabetic variant of $\bot_i$.*

Proof. *By induction on i. Let i be 0. In step 3 the head of $\bot_0$ is within the definite mode language of M (Definition 21) since every +type and -type place-marker is replaced by variables, every #type place-marker is replaced by ground terms and every variable has depth 0 (Definition 10). By construction the head a_h of the returned $\bot_0$ θ-subsumes a since inverting the one-one function hash gives a substitution from the variables in a_h to the terms in a. This substitution is most specific since every variable is replaced by a unique term. This proves the base case. Suppose that for all i up to and including k Algorithm 40 correctly constructs a most-specific clause $\bot_k$ such that $\bot_k$ is the most-specific clause in $\mathcal{L}_k(M)$ which θ-subsumes $\bot$. It is now shown that this implies the same will hold for $k+1$. Consider step 5 for $k+1$. The +type place-markers in the atom of m are replaced by variables of depth at most k which represent terms in InTerms. These terms must either have been placed in InTerms as +type in the head (step 3) or -type from step 5 at an earlier value of k. -type place-markers are replaced by variables of depth at most $k+1$ and #type by ground terms. Therefore $\bot_{k+1}$ is in $\mathcal{L}_{k+1}(M)$. Also by construction a_b subsumes an atom in the body of $\bot$ with substitution θ_b, and the substitution is most specific since all variables map to unique terms in $\bot$. $T(m)$ corresponds to all combinations of +type substitutions, which makes $\bot_{k+1}$ an alphabetic variant of the maximally specific clause in $\mathcal{L}_{k+1}(M)$ which θ-subsumes $\bot$. This proves the step and completes the proof.* $\Box$

The time-complexity of Algorithm 40 is proportional to the cardinality of $\bot_i$.

Theorem 26 Cardinality of $\bot_i$. *Let h, i, B, M be defined as in Definition 24 and let $|M|$ denote the cardinality of M. Let the number of +type and -type occurrences in each modeh in M be bounded by constants j^- and j^+ respectively. Let the number of +type and -type occurrences in each modeb in M be bounded by j^+ and j^- respectively. Let the recall of each m in M be bounded by the constant r. The cardinality of $\bot_i$ is bounded by $(r|M|j^+j^-)^{ij^+}$.*

Proof. *By induction. The clause $\bot_0$ contains only a head so its cardinality is 1. This proves the base case. Assume true for all i up to and including k and show for $i = k+1$. The number of terms associated with +type in the head or -type in the body of $\bot_k$ is $j^-(r|M|j^+j^-)^{kj^+}$. These can be used to replace j^+ +type place-markers in $|M|$ modeb declarations and the atom can be recalled r times, giving a cardinality of $\bot_{k+1}$ of at most $(r|M|j^+j^-)^{(k+1)j^+}$. This proves the step and completes the proof.* $\Box$

By default $i = 3$ in Progol and typically $j^+ \leq 2$. However, since in most cases relatively few atoms are true in the least Herbrand model of $B \wedge \overline{e}$ when $|M| < 10$ it is usually the case that $\bot_3$ has cardinality of less than 100 atoms.

9 Refinement

9.1 Refinement operators

When generalising an example e relative to background knowledge B, Progol constructs $\perp_i$ and searches from general to specific through the sub-lattice of single clause hypotheses H such that $\square \preceq H \preceq \perp_i$. This sub-lattice is bounded both above and below. The search is therefore better constrained than other general to specific searches, such as those in MIS [57] and FOIL [49], in which the sub-lattice being searched is not bounded below.

For the purposes of searching a lattice of clauses ordered by θ-subsumption Shapiro [57] introduced the concept of refinement operators. Suppose $\mathcal{L}$ is a (potentially infinite) set of clauses and C is an element of $\mathcal{L}$. Then the refinement operator ρ is defined such that $\rho(C) \subseteq \mathcal{L}$. ρ is said to be *sound* if and only if for each D in $\rho(C)$ it is the case that $C \preceq D$. Also $\rho^0(C) = \{C\}$ and $D \in \rho^i(C)$ if and only if there exists $D' \in \rho^{i-1}(C)$ and $D = D'$ or $D \in \rho(D')$. The closure $\rho^*(C)$ is $\rho^0(C) \cup \rho^1(C) \cup \ldots$

According to [21] ρ is *complete* if and only if for each D in $\mathcal{L}$ there is an alphabetic variant of D in $\rho^*(\square)$. ρ is *finite* if and only if for all $C \in \mathcal{L}$ the cardinality of $\rho(C)$ is finite. ρ is *proper* if and only if for each clause C and $D \in \rho(C)$ it is the case that $C \prec D$. It is shown in [20] that Shapiro's ρ is not complete. It is also shown that there does not exist ρ which is finite, proper and complete.

Redundancy of refinement operators is investigated in [12, 7]. The refinement operator ρ is redundant if and only if there exist clauses C, C', D in $\mathcal{L}$ such that $D \in \rho(C)$ and $D \in \rho(C')$ and C is not an alphabetic variant of C'. Since both MIS and FOIL employ redundant refinement operators, the same clause D can be reached repeatedly when applying ρ to various C and C'.

9.2 The refinement operator in Progol

The refinement operator in Progol is designed to avoid redundancy and to maintain the relationship $\square \preceq H \preceq \perp_i$ for each clause H.

Since $H \preceq \perp_i$, it is the case that there exists a substitution θ such that $H\theta \subseteq \perp_i$. Thus for each literal l in H there exists a literal l' in $\perp_i$ such that $l\theta = l'$. Clearly there is a uniquely defined subset $\perp_i(H)$ consisting of all l' in $\perp_i$ for which there exists l in H and $l\theta = l'$. A non-deterministic approach to choosing an arbitrary subset S' of a set S involves maintaining an index k. For each value of k between 1 and n, the cardinality of S, we decide whether to include the kth element of S in S'. Clearly, the set of all series of n choices corresponds to the set of all subsets of S. Also for each subset of S there is exactly one series of n choices. To avoid redundancy and maintain θ-subsumption of $\perp_i$ Progol's refinement operator maintains both k and θ.

Definition 27 Progol refinement operator. *Let h, i, B, e, M and $\bot_i$ be defined as in Definition 24 and let n be the cardinality of $\bot_i$. Let k be a natural number, $1 \leq k \leq n$. Let C be a clause in $\mathcal{L}_i(M)$ and θ be a substitution such that $C\theta \subseteq \bot_i$. Below a literal l corresponding to a mode m_l in M is denoted simply as $p(v_1, .., v_m)$ despite the sign of m_l and function symbols in $a(m_l)$. A variable is splittable if it corresponds to a +type or -type in a modeh or if it corresponds to a -type in a modeb. $\langle C', \theta', k'\rangle$ is in $\rho(\langle C, \theta, k\rangle)$ if and only if either*

1. *$C' = C \cup \{l\}$, $k' = k$, $\langle l, \theta'\rangle$ is in $\delta(\theta, k)$ and $C' \in \mathcal{L}_i(M)$ or*

2. *$C' = C$, $k' = k + 1$, $\theta' = \theta$ and $k < n$.*

$\langle p(v_1, .., v_m), \theta'\rangle$ is in $\delta(\theta, k)$ if and only if θ' is initialised to θ, $l_k = p(u_1, .., u_m)$ is the kth literal of $\bot_i$ and for each j, $1 \leq j \leq m$,

1. *if u_j is splittable then $v_j/u_j \in \theta'$ else $v_j/u_j \in \theta$ or*

2. *if u_j is splittable then v_j is a new variable not in dom(θ) and $\theta' = \theta \cup \{v_j/u_j\}$.*

In Definition 27 the variables in $\bot_i$ form a set of equivalences classes over the variables in any clause C which θ-subsumes $\bot_i$. Thus we could write the equivalence class of u in θ as $[v]_u$, the set of all variables in C such that v/u is in θ. The second choice in the definition of δ adds a new variable to an equivalence class $[v_j]_{u_j}$. This will be referred to as *splitting* the variable u_j. Note that in Definition 27 a variable is not splittable if it corresponds to a +type in a modeb since the resulting clause would violate the mode declaration language $\mathcal{L}(M)$ (see Definition 21). The following is an example of variable splitting.

Example 28 Applying ρ in list reversal. *Suppose M consists of the following mode delarations.*

modeh(,reverse(+list,-list))*	*modeb(*,+list=[-int|-list]*
modeb(,+any= #any)*	*modeb(*,reverse(+list,-list))*
modeb(,append(+list,[+int],-list))*	

The types and other background knowledge are defined as follows.

$$B = \begin{cases} any(Term) \leftarrow \\ list([]) \leftarrow \\ list([H|T]) \leftarrow list(T) \\ Term = Term \leftarrow \\ reverse([], []) \leftarrow \\ append([], X, X) \leftarrow \\ append([H|T], L1, [H|L2]) \leftarrow append(T, L1, L2) \end{cases}$$

Let $h = 30$ and $i = 3$ and let the example be as below.

$$e = reverse([1], [1]) \leftarrow$$

C'	θ'	k'
$reverse(D,E) \leftarrow$	$\{D/A, E/A\}$	1
$reverse(D,D) \leftarrow$	$\{D/A\}$	1
$\Box$	$\emptyset$	2

C'	θ'	k'
$reverse(D,E) \leftarrow D = [F\|G], reverse(G,G)$	θ	6
$reverse(D,E) \leftarrow D = [F\|G], reverse(G,H)$	$\theta \cup \{H/C\}$	6
$reverse(D,E) \leftarrow D = [F\|G]$	θ	7

Figure 3: Two applications of ρ.

In this case $\perp_i$ is as follows.

$$\perp_i = reverse(A,A) \;\leftarrow\; A = [1], A = [B|C], B = 1, C = [], \\ reverse(C,C), append(C,[B],A)$$

Let $\langle C', \theta', k'\rangle$ be in $\rho(\langle \Box, \emptyset, 1\rangle)$. Then all $\langle C', \theta', k'\rangle$ are shown in the first table in Figure 3. Suppose that $C = (reverse(D,E) \leftarrow D = [F|G])$, $\theta = \{D/A, E/A, F/B, G/C\}$, $k = 6$ and $\langle C', \theta', k'\rangle$ is in $\rho(\langle C, \theta, k\rangle)$. Then all $\langle C', \theta', k'\rangle$ are shown in the second table in Figure 3.

By analogy to Shapiro's ρ we can talk of the soundness of Progol's ρ.

Lemma 29 Soundness of Progol's ρ. *Let h, i, B, e, M and $\perp_i$ be defined as in Definition 24 and let n be the cardinality of $\perp_i$. Let k be a natural number, $1 \leq k \leq n$. Let C be a clause in $\mathcal{L}_i(M)$ and θ be a substitution such that $C\theta \subseteq \perp_i$. $\langle C', \theta', k'\rangle \in \rho(\langle C, \theta, k\rangle)$ only if $C'\theta' \subseteq \perp_i$ and $C' \in \mathcal{L}_i(M)$.*
Proof. *Suppose the lemma is false. In that case there exists $\langle C', \theta', k'\rangle \in \rho(\langle C, \theta, k\rangle)$ and either $C'\theta' \not\subseteq \perp_i$ or $C' \notin \mathcal{L}_i(M)$. But according to Definition 27, $C' \in \mathcal{L}_i(M)$ or $C' = C$, in which case also $C' \in \mathcal{L}_i(M)$. Thus it must be that $C'\theta' \not\subseteq \perp_i$ in which case $C' = C \cup \{l\}$ and $k' = k$ where $\langle l, \theta'\rangle$ is in $\delta(\theta, k)$. But then according to the definition of δ, $C'\theta' \subseteq \perp_i$ which contradicts the assumption and completes the proof.* $\Box$

As with Shapiro's refinement operator we can define the closure set for Progol's ρ. Let X, Y, Z stand for triples of the form $\langle C, \theta, k\rangle$. Then $\rho^0(X) = \{X\}$ and $Y \in \rho^i(X)$ if and only if there exists $Z \in \rho^{i-1}(X)$ and $Y = Z$ or $Y \in \rho(Z)$. The closure $\rho^*(X)$ is $\rho^0(X) \cup \rho^1(X) \cup ...$ The following example shows that Progol's ρ is not complete due to the choice of ordering of $\perp_i$.

Example 30 Incompleteness of search. *Let B contain definitions for decrementation (dec), addition (plus) and the clause $mult(0,X,0) \leftarrow$ with appropriate* *mode declarations M and let the example e be the clause* $mult(1,1,1) \leftarrow$. *Then* $\bot_i$ *is the clause*

$$\begin{aligned} mult(A,A,A) \;\; \leftarrow \;\; & dec(A,B), plus(A,B,A), plus(B,B,B), \\ & mult(A,B,B), mult(B,B,B). \end{aligned}$$

Given this ordering over $\bot_i$ *there will be no element of Progol's* ρ^* *containing the clause*

$$mult(U,V,W) \leftarrow dec(U,X), mult(X,V,Y), plus(Y,V,W).$$

9.3 Complexity of ρ

In order to analyse the complexity of ρ we introduce an incremental variant of the Bell number [19] from combinatorics. The mth Bell number is the number of ways that a set S of cardinality m can be partitioned into non-empty equivalence classes.

Lemma 31 Number of splits of a variable. *Suppose that* δ *in Definition 27 has arguments* θ, k *and that the kth literal of* $\bot_i$ *has m splittable occurrences of only one variable u. Suppose also that the cardinality of* $[v]_u$ *in* θ *is n. The number of variants of* θ' *is given by the function s as follows.*

$$s(n,m) = \begin{cases} 1 & \text{if } m = 0 \\ s(n,m\text{-}1)n+s(n+1,m\text{-}1) & \text{if } m > 0 \end{cases}$$

Proof. *If* $m = 0$ *there is only one substitution,* $\theta' = \theta$. *If* $m > 0$ *consider the first occurrence of u in* l_k. *In* δ *the choice can be to not split u (case 1) or to split u (case 2). In case 1, the set of* θ' *variants is* $\{\theta\}$ *crossed with the set of n choices for* v_1/u *crossed with the set of s(n,m-1) variants for the remaining m-1 occurrences of u in* l_k. *In case 2, if the new variable is v then the set of* θ' *variants is* $\{\theta\}$ *crossed with* $\{v/u\}$ *crossed with the set of s(n+1,m-1) variants for the remaining m-1 occurrences of u in* l_k. *This gives a total of s(n,m-1)n+s(n+1,m-1) variants of* θ'. □

A partial tabulation of the function s is shown in Figure 4.[5]

Remark 32 Bounds on s. *Let* n, m *be natural numbers.* $n^m \le s(n,m) \le (n+m)^m$.
Proof. *For* m=0, n^0=$s(n,0)$=$(n+0)^0$=1. *Consider s in terms of the recurrence* n^m=$n^{m-1}n$. *For all* $n \ge 0$ *and* $m>0$ *it is the case that* $s(n,m-1)n < s(n,m) < s(n+m,m-1)n+s(n+m,m-1)$. □

Example 33 *Suppose in Definition 27 that* $C = p(V) \leftarrow$ *and* $\theta = \{V/U\}$ *and* $l_k = q(U,U,U)$ *where the last two occurrences of U in* l_k *are -type. Then in Lemma 31 this gives m=2, n=1, and s(n,m)=5. The 5 variants of* $l_k\theta'$ *are* $q(V,V,V)$, $q(V,V,W)$, $q(V,W,V)$, $q(V,W,W)$ *and* $q(V,W,Z)$.

[5] The Bell function can be expressed simply as $B(m) = s(0,m)$.

	n	0	1	2	3	4	5	6	7
m									
0		1	1	1	1	1	1	1	1
1		1	2	3	4	5	6	7	
2		2	5	10	17	26			
3		5	15	37					
4		15							

Figure 4: A partial tabulation of the function s.

We are now in a position to give a function for the cardinality of ρ.

Theorem 34 The cardinality of ρ. *Let C, θ, k and l_k be as in Definition 27. Suppose that l_k contains p splittable variables and q non-splittable variables. Let m_x, $1 \leq x \leq p$, and m_y, $1 \leq y \leq q$, denote respectively the number of occurrences of v_x and v_y in the splittable and non-splittable variables of l_k. Let n_x, $1 \leq x \leq p$, and n_y, $1 \leq y \leq q$, denote respectively the number of u_x and u_y such that u_x/v_x and u_y/v_y are in θ. Then the cardinality of $\rho(\langle C, \theta, k\rangle)$ is*

$$|\rho(\langle C, \theta, k\rangle)| = (\Pi_{x=1}^{p} n_x^{m_x})(\Pi_{y=1}^{q} s(n_y, m_y)) + 1.$$

Proof. *In Definition 27, ρ chooses between 2 cases. Since the second choice produces a unique solution, the cardinality of ρ is one greater than the cardinality of the associated function δ. Only the first case of δ is applicable to non-splittable variables. Thus for each of the m_x occurrences of v_x in l_k there are n_x choices of u_x/v_x, giving $n_x^{m_x}$ variants. The set of all substitutions θ' for l_k is $\{\theta\}$ crossed with the set of variants for each v_x, $1 \leq x \leq p$ crossed with the set of variants for each v_y, $1 \leq y \leq q$. This gives a total of $(\Pi_{x=1}^{p} n_x^{m_x})(\Pi_{y=1}^{q} s(n_y, m_y))$ different substitutions θ' for the function δ and the same value plus 1 for the cardinality of ρ.* $\Box$

From Remark 32 it can be seen that $|\rho(\langle C, \theta, k\rangle)|$ is exponential in p, q, m_x and m_y. This reiterates the requirement indicated by Theorem 26 that for the sake of polynomial tractability p,m_x and q, m_y should be bounded respectively by constants j^+ and j^-.

In the implementation of ρ Progol simply decodes each of the natural numbers between 1 and $|\rho(\langle C, \theta, k\rangle)|$ into clauses and updates θ and k appropriately. The details of this decoding process are omitted.

10 Searching the subsumption lattice

To search the subsumption lattice Progol applies an A^*-like algorithm [45] to find a clause C, $\Box \preceq C \preceq \bot_i$, with maximal Occam compression (Definition 4).

The encoding measure is the total number of atom occurrences in a *reduced* logic program. Logic programs are reduced by eliminating *redundant* clauses.

Definition 35 Redundant clauses. *Let C be a clause and T be a set of clauses. C is redundant in $T \cup C$ if and only if $T \models C$.*

Definition 36 Reduced set of clauses. *Let T be a set of clauses. T is reduced iff T contains no redundant clauses.*

Progol's algorithm for finding C with maximal Occam compression is Algorithm 42 in Appendix D.2. The algorithm searches through the state space defined by elements of $\rho^*(\langle \Box, \emptyset, 1\rangle)$. A lookahead function h_s is used to increase efficiency when searching for 'variable-chaining' clauses. A clause is variable-chaining if and only if it contains a chain of variables $v_1, .., v_n$ such that v_1, v_n are +type and -type respectively in the head of C and each v_i, v_{i+1} are +type and -type respectively in an atom in the body of C. The recursive clause for reversing lists

$$\text{reverse}(A, B) \leftarrow A = [C|D], \text{reverse}(C, E), \text{append}(E, [A], B) \tag{5}$$

(see Example 28) is variable-chaining. A clause C is called I/O complete if and only if each -type variable in the head of C is found in the body of C. Clause (5) is I/O complete given the mode declarations in Example 28.

Lemma 37 Function h_s defines I/O complete lookahead. *Let $\perp_i$ and $s = \langle C, \theta, k\rangle$ be as in Definition 41 in Section D.2. For every I/O complete C' such that $s' = \langle C', \theta', k'\rangle \in \rho^*(\langle C, \theta, k\rangle)$ it is the case that $|C'| - |C| \geq h_s$.*
Proof. *By mathematical on induction on h_s. Suppose v is in the body of C, then $h_s = 0$ and the lemma holds in the base case. Suppose, by mathematical induction, that for all I/O complete C' and for all $s_d = \langle C_d, \theta_d, k_d\rangle$ for which $h_{s_d} = d$ it is the case that $|C'| - |C_d| \geq h_{s_d}$ and suppose that there exists such $s_d \in \rho(s)$. According to Definition 27 either $C_d = C$ and $\theta_d = \theta$ in which case for all I/O complete C' it is the case that $|C'| - |C| \geq h_s = d$ or else $C_d = C \cup \{l\}$ and $|C'| - |C| \geq (h_{s_d} + 1) \geq h_s$. This proves the step and completes the proof.* $\Box$

10.1 Correctness and time complexity

Note that in order to ensure polynomial tractability of Algorithm 42, the user is required to provide a bound c on the cardinality of the clause body.

Theorem 38 Correctness of Algorithm 42. *Let $E, h, i, B, e, M, \perp_i, c$ be as in Definition 41. Let $S = \rho^*(\langle \Box, \emptyset, 1\rangle)$ and S_c be the set of all elements s of S such that $c_s \leq c$. If $s = \langle C, \theta, k\rangle$ then $C(s) = C$. We say that clause C explains example e if and only if $B \wedge C \wedge \overline{e} \vdash_h \Box$ and $B \wedge C \wedge E \vdash_h \Box$. If S_c does not contain any s such that $C(s)$ explains e and $f_s > 0$ then Algorithm 42 returns 'no compression'. Otherwise Algorithm 42 returns $s \in S_c$ such that $C(s)$ explains*

e and there does not exist $s' \in S_c$ for which $C(s')$ explains e and $f_{s'} > f_s$.

Proof. *By contradiction. Assume the theorem is false. Then either (a) the algorithm does not terminate or (b) there exists $s \in S_c$ such that $C(s)$ explains e, $f_s > 0$ and 'no compression' is returned or (c) s is returned and either $C(s)$ does not explain e or $f_s \leq 0$ or (d) s is returned and $C(s)$ explains e and $f_s > 0$ but there exists $s' \in S_c$ for which $C(s')$ explains e and $f_{s'} > f_s$.*

First consider (a). Since ρ (Definition 27) either adds another literal or moves forward by one through $\bot_i$, there can only be a finite number of elements of $s \in S_c$. In each cycle at least one of these, say s, is transferred from Open to Closed in steps 3 and 4 and never reappears in Open again due to the construction in step 6. Open will never contain elements other than those in S_c due to the third condition in the predicate prune. Thus there are only a finite number of cycles and each operation terminates in finite time. This refutes (a).

Therefore instead suppose (b) there exists $s \in S_c$ such that $C(s)$ explains e, $f_s > 0$ and 'no compression' is returned in step 8. But step 8 can only be entered after step 7, in which case if Open $= \emptyset$ then terminated must have been false and therefore Closed contained no s for which $C(s)$ explained e and $f_s > 0$. But if there exists $s \in S_c$ for which $C(s)$ explains e and $f_s > 0$ then there must be $s' \in S_c$ for which prune(s') was true, since otherwise s would eventually have been transferred to Closed. But the first condition of prune could not have been true of s' since otherwise at worst s' would have succeeded as best in terminated. The second condition of prune could not have been true of s' since if $g_{s'} \leq 0$ then also $g_s \leq 0$ and thus $f_s \leq 0$. The third condition of prune could not be true either since if $c_{s'} \geq c$ then either $C(s') = C(s)$ or $C(s) \notin S_c$. This refutes (b).

Instead suppose (c) s is returned and either $C(s)$ does not explain e or $f_s \leq 0$. But if s is returned in step 7 then terminated must be true in which case $n_s = 0$ and $f_s > 0$. For all $s \in S$, by the construction of $\bot_i$ (Definition 24) and the soundness of ρ (Definition 29) $B \wedge C(s) \wedge \overline{e} \vdash_h \Box$. Also since $n_s = 0$ it follows that $B \wedge C(s) \wedge E \not\vdash_h \Box$. Therefore $C(s)$ explains e and $f(s) > 0$. This refutes (c).

Lastly suppose (d) s is returned and $C(s)$ explains e and $f_s > 0$ but there exists $s' \in S_c$ for which $C(s')$ explains e and $f_{s'} > f_s$. But s' cannot be in Closed since s =best(Closed) and therefore $f_s \geq f'_s$. Therefore on return from step 7 there must exist s'' in Open for which $s' \in \rho^*(s'')$. But in that case according to the terminated predicate $f_s \geq g_{s''} \geq g_{s'} \geq f_{s'}$. This refutes (d) and completes the proof. $\Box$

In the worst case Algorithm 42 will consider all elements of S_c in Theorem 38.

Theorem 39 Cardinality of S_c. *Let $i, \bot_i, S_c, c$ be as in Definition 41. Let j^+, j^- be as in Theorem 26 and let $j = j^+ + j^-$. Let $|S|$ denote the cardinality of any set S. $|S_c| \leq |\bot_i|^{c+1} j(c+1)^j$.*

Proof. *The elements $s = \langle C, \theta, k \rangle$ of S_c are all those $s \in \rho^*(\langle \Box, \emptyset, 1 \rangle)$ for which $|C| \leq (c+1)$. Since $C\theta \subseteq \bot_i$ we can view the construction of s as the choice* *(with possible repeats) of $c+1$ elements from $\bot_i$ followed by the choice of θ. It is simplest to treat $C\theta$ (with repeat literals) as though it were a single atom and use the bounds in Remark 32 to calculate the worst case for the number of variants of θ. In this case there are at most $|\bot_i|^{c+1}$ ways of choosing the elements of $C\theta$ and $j(c+1)^j$ ways of choosing θ. Thus $|S_c| \leq |\bot_i|^{c+1} j(c+1)^j$.* $\Box$

From Theorem 26 and 39 we find that $|S_c|$ is of order $O(r|M|^{2ij(c+1)})$. Clearly, for tractability i, j, c must be small constants.

10.2 Cover set algorithm

Progol uses a simple cover set algorithm much like that employed in Michalski's AQ family of algorithms [28]. It repeatedly generalises examples in the order found in the Progol source file and adds the generalisation to the background knowledge. Examples which are redundant relative to the background knowledge are then removed (redundancy is based on Definition 35). The cover set algorithm is given in Appendix D.3. Clearly Algorithm D.3 terminates in at most $|E|$ iterations.

Note that each clause is unflattened before being added to the background knowledge. If, as in Prolog, equality is assumed to be completely defined using only the axiom of identity ($\forall x.(x = x)$) then unflattening has no effect on the Herbrand models of a logic program. However, it does improve its readability. For instance, clause (5) in Section 10 can be unflattened to the following simpler clause.

$$\mathrm{reverse}([A|B], C) \leftarrow \mathrm{reverse}(B, D), \mathrm{append}(D, [A], C).$$

Note that the use of modeb declarations for '=' in Example 28 followed by the use of unflattening in Algorithm D.3 allows Progol to search through the term structure of hypothesised clauses. This is despite the fact that Progol's refinement operator (Definition 27) considers only variable/variable substitutions which map hypothesised clauses to subsets of $\bot_i$.

11 The Progol system

Progol was written in C by the author of this paper. Progol version 4.1 source code, example files and manual pages are freely available (for academic research) by anonymous ftp from ftp.comlab.ox.ac.uk in directory pub/Packages/ILP/progol4.1.

The design methodology for Progol was to present the user with a standard Prolog interpreter augmented with inductive capabilities. The syntax for examples, background knowledge and hypotheses is Dec-10 Prolog, with the usual augmentable set of prefix, postfix and infix operators. Headless Horn clauses, representing constraints are used to represent negative examples and constraints.

These are stored internally as clauses with head 'false'. Thus the following statement can be placed in the Progol source file.

$$: -black(X), white(X).$$

This is stored internally as the following definite clause.

$$false : -black(X), white(X).$$

In this way both the testing of negative examples and of general constraints reduces to seeing whether 'false' is provable. Headless clause constraints can be learned from ground headless unit clauses by use of a modeh for the predicate 'false'. An example of this can be found in the Progol4.1 distribution dataset 'animals.pl'.

The standard library of primitive predicates described in Clocksin and Mellish [3] is built into Progol and available as background knowledge. Thus the following command-line can be given to Progol when using the infix predicate '=<' for learning ranges of integers.

|− modeh(1,p(+int)), modeb(3,#int =< +int), modeb(3,+int =< #int)?

The Progol prompt is |− and int is a built-in single arity predicate which is true for all integers. Note that Progol queries are terminated by '?' rather than the usual '.' in Prolog. This allows queries to be distinguished from assertions. Assertions terminated by '.' can also be made at the Progol prompt level. The user can request examples to be generalised from the prompt by terminating the example clause by a '!'. Unless the predicate 'search' is executed first, a '!' statement will simply show the user the clause $\perp_i$ for the example. Thus the mode declarations above will allow the following interaction.

|− p(5)!
[Most specific clause is]
p(A) :- 3=<A, 4=<A, 5=<A, A=<5, A=<6, A=<7.

In this $\perp_3$ clause the modeb declarations (given above) for '=<' are used. In step 5 of Algorithm 40 the goals $X =< 5$ and $5 =< Y$ are both recalled 3 times and succeed with substitutions 3,4,5 for X and 5,6,7 for Y. The #int place-markers are replaced by 3,4,5 and 5,6,7 respectively and the +int place-marker is replaced by the unique variable A using the hash function described in Algorithm 40.

Although Progol can be used interactively, it is often more convenient to run it in batch mode. In this case, when called from the operating system shell, Progol is given the name of the example file as an argument. Progol then simply generalises every predicate for which a modeh is declared and shows the results as output.

Progol can learn ranges and functions with numeric data. These can be either integer or floating point by simply making use of the built-in predicates 'is', '<',

'=<', etc. This is best exemplified in the Progol4.1 dataset order4, in which qualitative regression is applied in conjecturing Newton's inverse square law from artificial floating point data.

The choice of engineering a complete Prolog interpreter was taken in order to make induction a first-class and efficient operation on the same footing as deductive theorem proving. This allows implementation of low-level operations such as depth-bounding of the theorem prover and rapid virtual assertion and retraction of clauses into the clause set.

12 Results

Results of a series of experiments involving Progol in learning to predict mutagenic molecules can be found in [58, 59, 60]. A description of Progol doing qualitative regression can be found in [41]. Qualitative regression is carried out by using mode declarations to define a family of 3 different functions (linear, polynomial in one term and exponential) and using these in competition to fit the data. The equation solver is supplied as user-defined background knowledge.

Appendix E gives a table of runtimes on a SPARCstation 10 for learning the various examples in the distribution version of Progol4.1. The numbers of clauses in E^+, E^-, B and H are also given for each dataset. Note that the datasets 'animals', 'exp', 'family' and 'set' involve learning a series of related predicates. These runtimes are comparable with those of FOIL [49], despite the fact that FOIL does incomplete heuristic search to find clauses. FOIL also uses extensional background knowledge rather than the intensional background knowledge of Progol.

13 Conclusion

This paper traces the line of development followed by the author in investigating induction as the inverse of deduction. It has been shown that the idea of inverting resolution proofs used in Duce and Cigol can be greatly simplified by considering this as a special case of inversion of entailment. However, the notion of inverting entailment is of a more fundamental nature than that of inverting proof, since it is based on the model-theory which underlies proof. This approach has led to the development of a new state-of-the-art ILP system called Progol, which is available for academic research purposes by anonymous ftp (see Section 11). For each example Progol develops a most specific clause $\perp_i$ within the user-defined mode language, and uses this to guide an A^*-like search through clauses which subsume $\perp_i$. Each invocation of the search returns a clause which is guaranteed to maximally compress the data. Despite the admissibility of this search, the learning times in Appendix E are comparable with FOIL, an algorithm which carries out a truncated heuristic search and allows only extensional background knowledge.

Figure 2 in Section 7 shows various ways in which Progol could be made more powerful. At present Progol can only deal effectively with the first and third form of $\perp$. If Progol could prove not only positive ground facts but also negative ones then it would be possible to construct $\perp$ in the form of the second entry in Figure 2. This would have applications in theory revision. However, for the purposes of theory revision, Progol would need to have a strategy for specialising over-general clauses. The construction of sub-saturants (Section 6.1) would allow Progol to find all generalisations of recursive clauses, such as the one in the fourth entry of Figure 2. Both the second and fourth form of generalisation in Figure 2 will lead to multiple definite $\perp$ clauses. Dealing with the multiplicity of $\perp$ clauses will require improvements in Progol's search techniques. The incompleteness of the present search (see Example 30) also needs to be addressed.

Definition 9 suggests a way in which Progol could be made to learn effectively when provided with only positive example data. This would have real world applications in areas such as natural language learning, in which it is common to find positive-only data sources.

No learnability results have yet been shown for Progol. U-learnability (Appendix B) offers a promising direction for such results.

The author believes that inverse entailment offers many new avenues in the rapidly maturing research area of Inductive Logic Programming.

Acknowledgements

Many thanks are due to my wife, Thirza Castello-Cortes, who has not only shown super-human tolerance during the long incubation and writing of this paper but has also helped by proof-reading various versions. The author would also like to thank Donald Gillies for pointing out the foundational (but almost wholly disregarded) work of Stanley Jevons. Thanks are also due to David Page, and Donald Michie for their helpful discussions and advice and to Ashwin Srinivasan, who produced the initial Prolog version of Progol. Valuable suggestions concerning the U-learnability model were given by Tony Hoare, Bill McColl, Michael Kearns and Paul Vitanyi. This work was supported partly by the Esprit Basic Research Action ILP (project 6020), EPSRC grant GR/J46623 on Experimental Application and Development of ILP and an EPSRC Advanced Research Fellowship held by the author. The author is supported by a non-stipendiary Research Fellowship at Wolfson College Oxford.

References

[1] M. Bain and S. Muggleton. Non-monotonic learning. In D. Michie, editor, *Machine Intelligence 12*. Oxford University Press, 1991.

[2] I. Bratko, S. Muggleton, and A. Varsek. Learning qualitative models of dynamic systems. In *Proceedings of the Eighth International Machine Learning Workshop*, San Mateo, Ca, 1991. Morgan-Kaufmann.

[3] W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, 1981.

[4] W. Cohen. Learnability of restricted logic programs. In S. Muggleton, editor, *Proceedings of the 3rd International Workshop on Inductive Logic Programming (Technical report IJS-DP-6707 of the Josef Stefan Institute, Ljubljana, Slovenia)*, pages 41–72, 1993.

[5] D. Conklin and I. Witten. Complexity-based induction. Technical report, Dept. of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, 1992.

[6] B. Dolsak and S. Muggleton. The application of Inductive Logic Programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, London, 1992.

[7] R. Dormer. *An Inductive Logic Programming Implementation*. PhD thesis, Oxford University Computing Laboratory, Oxford, 1993.

[8] C. Feng. Inducing temporal fault dignostic rules from a qualitative model. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, London, 1992.

[9] D.A. Gillies. Confirmation theory and machine learning. In *Proceedings of the Second Inductive Logic Programming Workshop*, Tokyo, 1992. ICOT TM-1182.

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

[11] G. Gottlob. Subsumption and implication. *Information Processing Letters*, 24(2):109–111, 1987.

[12] M. Grobelnik. Markus - an optimized model inference system. In *Proceedings of the ECAI workshop on Logical Approaches to Machine Learning*, 1992.

[13] P. Idestam-Almquist. Learning missing clauses by inverse resolution. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 610–617, Tokyo, 1992. ICOT.

[14] P. Idestam-Almquist. *Generalisation of Clauses*. PhD thesis, Stockholm University, 1993.

[15] W.S. Jevons. On the mechanisation of deductive inference. *Philosophical Transactions of the Royal Society of London*, 160:497–518, 1870.

[16] W.S. Jevons. *The Principles of Science: a Treatise on Logic and Scientific Method*. Macmillan, London, 1874.

[17] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2, 1992.

[18] R. King, S. Muggleton R. Lewis, and M. Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 89(23), 1992.

[19] V. Krishnamurthy. *Combinatorics: theory and applications*. Ellis Horwood, Chichester, England, 1986.

[20] P.R. van der Laag and Nienhuys-Cheng. Subsumption and refinement in model inference. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 95–114. Springer-Verlag, 1993.

[21] P.R. van der Laag and Nienhuys-Cheng. Existence and nonexistence of complete refinement operators. In Bergadano F. and De Raedt L., editors, *Proceedings of the 7th European Conference on Machine Learning*, volume 784 of *Lecture Notes in Artificial Intelligence*, pages 307–322. Springer-Verlag, 1994.

[22] S. Lapointe and S. Matwin. Sub-unification: a tool for efficient induction of recursive programs. In *Proceedings of the Ninth International Machine Learning Conference*, Los Altos, 1992. Morgan Kaufmann.

[23] C. Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms*. PhD thesis, University of California, Berkeley, 1967.

[24] M. Li and P. Vitanyi. *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag, Berlin, 1993.

[25] C.X. Ling. Learning the past tense of english verbs: the symbolic pattern associators vs. connectionist models. *Journal of Artificial Intelligence Research*, 1:209–229, 1994.

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

[27] B. Meltzer. Power amplification for automatic theorem proving. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 165–179. Edinburgh University Press, Edinburgh, 1969.

[28] R. Michalski and J. Larson. Incremental generation of vl1 hypotheses: the underlying methodology and the description of program AQ11. ISG 83-5, Computer Science Department, Univ. of Illinois at Urbana-Champaign, 1980.

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

[30] S. 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.

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

[32] S. Muggleton. Inverting the resolution principle. In *Machine Intellience 12*. Oxford University Press, 1991.

[33] S. Muggleton. Inverting implication. In *Proceedings of the Second Inductive Logic Programming Workshop*, Tokyo, 1992. ICOT (Technical report TM-1182).

[34] S. Muggleton. Bayesian inductive logic programming. In W. Cohen and H. Hirsh, editors, *Proceedings of the Eleventh International Machine Learning Conference*, pages 371–379, San Mateo, CA, 1994. Morgan-Kaufmann.

[35] S. Muggleton. Inductive logic programming: derivations, successes and shortcomings. *SIGART Bulletin*, 5(1):5–11, 1994.

[36] S. Muggleton. Predicate invention and utilization. *Journal of Experimental and Theoretical Artificial Intelligence*, 6(1):127–130, 1994.

[37] S. 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.

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

[39] S. Muggleton, R. King, and M. Sternberg. Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657, 1992.

[40] S. Muggleton and C.D. Page. Self-saturation of definite clauses. In S. Wrobel, editor, *Proceedings of the Fourth International Inductive Logic Programming Workshop*, pages 161–174. Gesellschaft fur Mathematik und Datenverarbeitung MBH, 1994. GMD-Studien Nr 237.

[41] S. Muggleton and D. Page. Beyond first-order learning: inductive learning with higher-order logic. Technical Report PRG-TR-13-94, Oxford University Computing Laboratory, Oxford, 1994.

[42] S. Muggleton and D. Page. A learnability model for universal representations. Technical Report PRG-TR-3-94, Oxford University Computing Laboratory, Oxford, 1994.

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

[44] S. Muggleton, A. Srinivasan, and M. Bain. Compression, significance and accuracy. In D. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Machine Learning Conference*, pages 338–347, San Mateo, CA, 1992. Morgan-Kaufmann.

[45] N.J. Nilsson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, CA, 1980.

[46] G.D. Plotkin. A note on inductive generalisation. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, Edinburgh, 1969.

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

[48] R.J. Popplestone. An experiment in automatic induction. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 203–215. Edinburgh University Press, Edinburgh, 1969.

[49] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[50] J.R. Quinlan. Past tenses of verbs and first-order learning. In Zhang C., J. Debenham, and Lukose D., editors, *Proceedings of the 7th Australian Joint Conference on Artificial Intelligence*, pages 13–20, Singapore, 1993. World Scientific.

[51] J.C. Reynolds. Transformational systems and the algebraic structure of atomic formulas. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 135–151. Edinburgh University Press, Edinburgh, 1969.

[52] J. Rissanen. Modeling by Shortest Data Description. *Automatica*, 14:465–471, 1978.

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

[54] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, London, 1992.

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

[56] C.E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, 1963.

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

[58] A. Srinivasan, S.H. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: Ilp experiments in a non-determinate biological domain. In S. Wrobel, editor, *Proceedings of the Fourth International Inductive Logic Programming Workshop*. Gesellschaft fur Mathematik und Datenverarbeitung MBH, 1994. GMD-Studien Nr 237.

[59] A. Srinivasan, S.H. Muggleton, R.D. King, and M.J.E. Sternberg. The effect of background knowledge in inductive logic programming: a case study. Technical Report PRG-TR-9-95, Oxford University Computing Laboratory, Oxford, 1995.

[60] A. Srinivasan, S.H. Muggleton, R.D. King, and M.J.E. Sternberg. Theories for mutagenicity: a study of first-order and feature based induction. Technical Report PRG-TR-8-95, Oxford University Computing Laboratory, Oxford, 1995.

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

Appendixes

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 letters and digits. 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. As in Prolog, integers, '[]' and '.' are function symbols and if $t_1, t_2, \ldots$ are terms then '.'(t_1, t_2) can equivalently be denoted $[t_1|t_2]$ and '.'$(t_1,$'.'$(t_2, ..$'.'$(t_n, [])..))$ can equivalently be denoted $[t_1, t_2, .., t_n]$. A predicate symbol immediately followed by a bracketed n-tuple of terms is called an atomic formula, or atom. Every atom is a well-formed formula (wff). If W and W' are wffs then $\overline{W}$ (not W), $W \wedge W'$ (W and W'), $W \vee W'$ (W or W') and $W \leftarrow W'$ (W implied by W') are wffs. $W \wedge W'$ is a conjunction and $W \vee W'$ is a disjunction. If v is a variable and W is a wff then $\forall v.W$ (for all v W) and $\exists v.W$ (there exists a v such that W) are wffs. v is said to be universally quantified in $\forall v.W$ and existentially quantified in $\exists v.W$. The wff W is said to be function-free if and only if W contains no function symbols. Both A and $\overline{A}$ are literals whenever A is an atom. In this case A is called a positive literal and $\overline{A}$ is called a negative literal. A set of literals is called a clause. The empty clause is represented by $\square$. A clause represents the disjunction of its literals. Thus the clause $\{a_1, a_2, ..\overline{a_i}, \overline{a_{i+1}}, .., \overline{a_n}\}$ can be equivalently represented as $(a_1 \vee a_2 \vee .. \overline{a_i} \vee \overline{a_{i+1}} \vee .. \vee \overline{a_n})$ or $a_1; a_2; .. \leftarrow a_i, a_{i+1}, .., a_n$. All the variables in a clause are implicitly universally quantified. A Horn clause is a clause which contains at most one positive literal. A definite clause is a clause which contains exactly one positive literal. A positive literal in either a Horn clause or definite clause is called the head of the clause while the negative literals are collectively called the body of the clause. A set of clauses in which no pair of clauses share a common variable 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 $\{C_1, C_2, .., C_n\}$ can be equivalently represented as $(C_1 \wedge C_2 \wedge .. \wedge C_n)$. Every clausal theory is said to be in clause-normal form. Every wff can be transformed to an equivalent wff in clause normal form. If $C = \forall l_1 \vee ..l_n$ is a clause then $\overline{C} = \exists l_1 \wedge .. \wedge l_n$. In this case $\overline{C}$ is not in clause normal form since the variables are existentially quantified. $\overline{C}$ can be put in clause normal form by substituting each occurrence of every variable in $\overline{C}$ by a unique constant not found in C. The process of replacing (existential) variables by constants is called skolemisation. The unique constants are called skolem constants. 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. 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 Substitutions and models

Let $\theta = \{v_1/t_1, .., v_n/t_n\}$. θ 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 θ, or dom(θ), and $\{t_1, .., t_n\}$ the range of θ, or rng(θ). Lower-case Greek letters are used to denote substitutions. Let E be a wff or a term and $\theta = \{v_1/t_1, .., v_n/t_n\}$ be a substitution. The instantiation of E by θ, written $E\theta$, is formed by replacing every occurrence of v_i in E by t_i. Atom a θ-subsumes atom b, or $a \preceq b$ if and only if there exists a substitution θ such that $a\theta = b$. Clause C θ-subsumes clause D, or $C \preceq D$ if and only if there exists a substitution θ such that $C\theta \subseteq D$. The Herbrand universe of the wff W is the set of all ground terms composed of function symbols found in W. The Herbrand base of the wff W is the set of all ground atoms composed of predicate and function symbols found in W. An interpretation is a total function from ground atoms to $\{\square, \blacksquare\}$. A Herbrand interpretation I of wff W is an interpretation whose domain is the Herbrand base of W. I can equivalently be represented as a subset of the atoms a in the Herbrand base of W for which $I(a) = \blacksquare$. Below all interpretations I are assumed to be Herbrand. The atom a is true in I if $I(a) = \blacksquare$ and false otherwise. The wff $\overline{W}$ is true in I if W is false in I and is false otherwise. The wff $W \wedge W'$ is true in I if both W and W' are true in I and false otherwise. The wff $W \vee W'$ is true in I if either W or W' is true in I and false otherwise. The wff $W \leftarrow W'$ is true in I if $W \vee \overline{W'}$ is true in I and false otherwise. If v is a variable and W is a wff then $\forall v.W$ is true in I if for every term t in the Herbrand universe of W the wff $W\{v/t\}$ is true in I. Otherwise $\forall v.W$ is false in I. If v is a variable and W is a wff then $\exists v.W$ is true in I if $\overline{\forall v.\overline{W}}$ is true in I and false otherwise. Interpretation M is a model of wff W if and only if W is true in M. A wff W is satisfiable if there exists a model of W and unsatisfiable otherwise. Consequently W is unsatisfiable if and only if $W \models \square$. Herbrand's theorem states that a wff W is satisfiable if and only if W has a Herbrand model. Every logic program P has a unique least Herbrand model M such that M is a model of P and every atom a is true in M only if it is true in all Herbrand models of P. Let W and W' be two wffs. We say that W semantically entails W', or $W \models W'$ if and only if every model of W is a model of W'. Let X, Y and Z be wffs. Then according to the Deduction theorem $X \wedge Y \models Z$ if and only if $X \models \overline{Y} \vee Z$. Let $\frac{X}{Y}$ be an inference rule. Then $\frac{X}{Y}$ is said to be sound if and only if $X \models Y$. Suppose I is a set of inference rules containing $\frac{X}{Y}$ and W, W' are wffs. Then $W \vdash_I W'$ if W' is formed by replacing an occurrence of X in W by Y. Otherwise $W \vdash_I W'$ if $W \vdash_I W''$ and $W'' \vdash_I W'$. We say that W syntactically entails W' using inference rules I, if and only if $W \vdash_I W'$. The set of inference rules I is said to be deductively sound and complete if and only if each rule in I is sound and $W \vdash_I W'$ whenever $W \models W'$. Let W and W' be two wffs. We say that W is more general than W' (conversely W' is more specific than W) if and only if $W \models W'$.

A.3 Resolution

The substitution θ is said to be a variable renaming if and only if each $u \in \mathrm{dom}(\theta)$ and $v \in \mathrm{rng}(\theta)$ are variables. Let W and W' be two wffs. If there exists a variable renaming θ such that $W\theta = W'$ then W, W' are said to be alphabetic variants of each other. Wffs W, W' are said to be standardised apart if and only if there exists a variable renaming $\theta = \{u_1/v_1, ..u_n/v_n\}$, $\mathrm{vars}(W) \subseteq \mathrm{vars}(\theta)$ and $W\theta = W'$. The substitution θ is said to be the unifier of the atoms a and a' whenever $a\theta = a'\theta$. μ is the most general unifier (mgu) of a and a' if and only if for all unifiers γ of a and a' there exists a substitution δ such that $(a\mu)\delta = a\gamma$. Let C and D be clauses and a be an atom. The sound inference rule

$$\frac{C \vee a \qquad\qquad D \vee \overline{a}}{C \vee D}$$

is called resolution. $(C \cup D)\theta$ is said to be the resolvent of the clauses $C \cup \{a\}$ and $D \cup \{a'\}$ whenever C and D are standardised apart and θ is the mgu of the atoms a and $\overline{a'}$. Let T be a clausal theory. Robinson [53] 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 ($\Box$).

B Hypotheses, probabilities and U-learnability

B.1 U-learnability

The following is a variant of the U-learnability framework presented in [34, 42]. The teacher starts by choosing distributions F and G from the family of distributions $\mathcal{F}$ and $\mathcal{G}$ over concept descriptions $\mathcal{H}$ (wffs with associated bounds for time taken to test entailment) and instances X (ground wffs) respectively. The teacher uses F and G to carry out an infinite series of teaching sessions. In each session a target theory T is chosen from F. Each T is used to provide labels from $\{\blacksquare, \Box\}$ (True, False) for a set of instances randomly chosen according to distribution G. The teacher labels each instance x_i in the series $\langle x_1, .., x_m \rangle$ with $\blacksquare$ if $T \models x_i$ and $\Box$ otherwise. An hypothesis $H \in \mathcal{H}$ is said to explain a set of examples E whenever it both entails and is consistent with E. On the basis of the

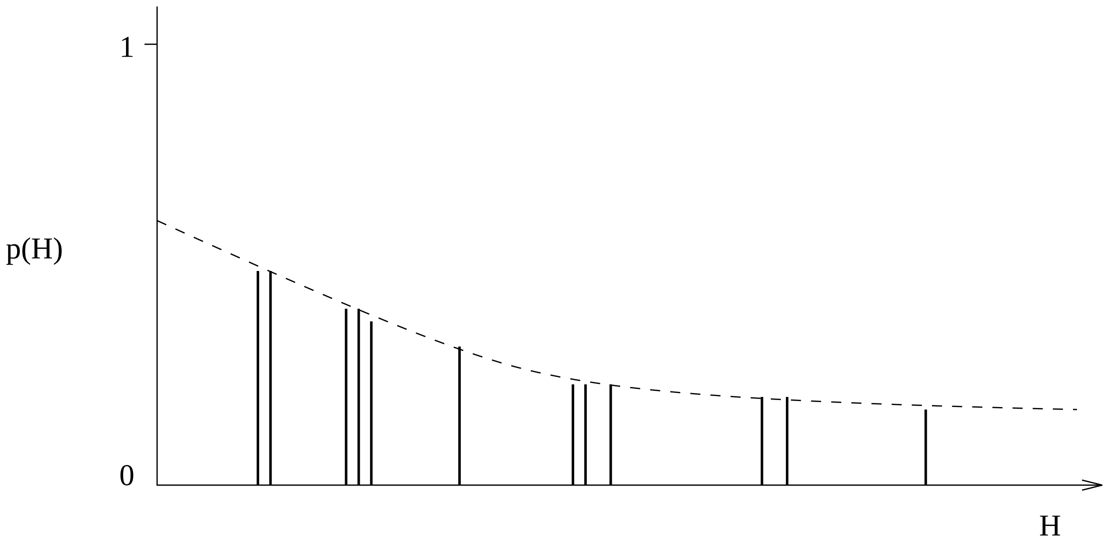

Figure 5: Prior and posterior probabilities of hypotheses.

series of labelled instances $\langle e_1, e_2, .., e_m \rangle$, a Turing machine learner L produces a sequence of hypotheses $\langle H_1, H_2, ..H_m \rangle$ such that $H_i \in \mathcal{H}$ explains $\{e_1, .., e_i\}$. H_i must be suggested by L in expected time bounded by a fixed polynomial function of i. The teacher stops a session once the learner suggests hypothesis H_m with expected error less than ϵ for the label of any x_{m+1} chosen randomly from G. $\langle F, G \rangle$ is said to be U-learnable if and only if there exists a Turing machine learner L such that for any choice of δ and ϵ $(0 < \delta, \epsilon \leq 1)$ with probability at least $(1 - \delta)$ in any of the sessions m is less than a fixed polynomial function of $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$.

B.2 Bayesian interpretation of setting

Figure 5 shows the effect $E = \{e_1, ..e_i\}$ has on the probabilities associated with hypotheses in $\mathcal{H}$. The learner's hypothesis language $\mathcal{H}$ is laid out along the X-axis with prior probability $p(H) = F(H)$ for H in $\mathcal{H}$ measured along the Y-axis, where

$$\sum_{H \in \mathcal{H}} p(H) = 1.$$

The descending dotted line in Figure 5 represents a bound on the prior probabilities of hypotheses before consideration of examples E. The hypotheses $\mathcal{H}_E$ $(\mathcal{H}_E \subseteq \mathcal{H})$ which explain E are marked as vertical bars. The prior probability of E, $p(E)$, is simply the sum of probabilities of hypotheses in $\mathcal{H}_E$. The conditional probability $p(E|H)$ is 1 in the case that that H explains E and 0 otherwise. The posterior probability of H is now given by Bayes theorem as

$$p(H|E) = \frac{p(H)p(E|H)}{p(E)}$$

With reference to Figure 5, for an hypotheses H which explains all the data, $p(H|E)$ will increase monotonically with increasing E. Also for two different hypotheses H_1, H_2 which explain E the following holds.

$$\frac{p(H_1|E)}{p(H_2|E)} = \frac{p(H_1)}{p(H_2)} \tag{6}$$

C Subsumption and least general generalisation

In the late 1960's the success of Robinson's [53] resolution procedure produced considerable interest in the problem of inducing first-order formulae. Both Meltzer [27] and Popplestone [48] carried out initial investigations into generalisation of ground formulae by replacement of constants with variables. In implementing his approach Meltzer decided to bound the number of resolutions involved in checking any hypothesis against examples. This was an important innovation which is now being used within Progol (Section 11).

In an alternative approach Reynolds [51] and Plotkin [46] investigated the problem of finding least general generalisations (lggs) of atoms. According to Plotkin [47],

> The work started with a suggestion by R.J. Popplestone (private communication) that, just as the unification algorithm was fundamental to deduction, so might a converse be of use in induction.

The relationship of lgg to unification is depicted in Figure 6. Atom g is a common generalisation of atoms a and b if an only if there exist substitutions ${\alpha_g}'$ and ${\beta_g}'$ such that $a = g{\alpha_g}'$ and $b = g{\beta_g}'$. The atom $lgg(a,b)$ is the least general generalisation of a and b if and only if $lgg(a,b)$ is a common generalisation of a and b and for each common generalisation g of a and b there exists a substitution δ_g such that $lgg(a,b) = g\delta_g$. The common instance i and most general instance are similarly defined for a and b (see Figure 6). In the case of the most general instance i of a and b Robinson [53] calls $\alpha_i\beta_i$ the most general unifier of a and b. Robinson describes an algorithm for constructing the most-general unifier of two atoms. Robinson's unification algorithm is the basis of resolution theorem proving. Plotkin and Reynolds describe an efficient algorithm for computing the least general generalisation of two atoms.

$[a]$ is the equivalence class of all atoms which are variable renamings of a. Reynolds showed that the set of all equivalence classes of atoms augmented by the symbols $\top$ and $\perp$ form a non-modular lattice. Thus, $[a] \sqcap [b] = [lgg(a,b)]$ and $[a] \sqcup [b] = [mgi(a,b)]$, where $\sqcap$ and $\sqcup$ are both commutative and associative, though neither distributes over the other.

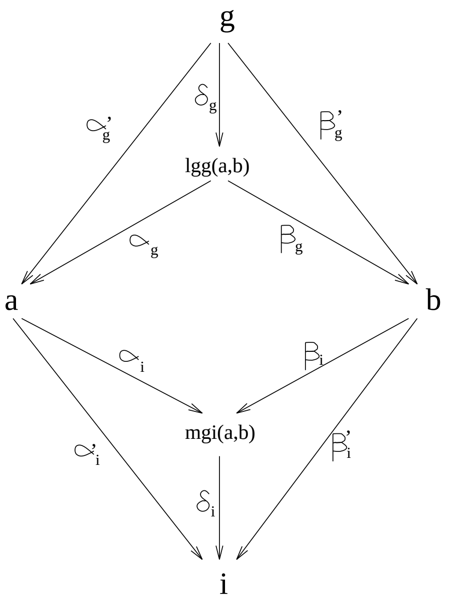

Figure 6: Relationship of lgg and mgi.

In [46] Plotkin extended the investigation to clauses ordered by θ-subsumption. Clause C θ-subsumes clause D, or $C \preceq D$ if and only if there exists a substitution θ such that $C\theta \subseteq D$. Just as with atoms, clause G and I are respectively a common generalisation and a common instance of C and D if and only if $G \preceq C, D$ and $C, D \preceq I$. For clauses C and D there is a least general generalisation $lgg(C, D)$ and most general instance $mgi(C, D)$, both unique up to renaming, such that for every common generalisation G and common instance I of C and D it is the case that $G \preceq lgg(C, D)$ and $mgi(C, D) \preceq I$. The cardinality of the least general generalisation of two clauses is bounded by the product of the cardinalities of the two clauses.

Plotkin [46] went on to define the lgg of two clauses relative to clausal background knowledge B. The relative least general generalisation of clauses ($rlgg_B$) is potentially infinite for arbitrary B. When B consists of ground unit clauses only the $rlgg_B$ of two clauses is finite. However the cardinality of the $rlgg_B$ of m clauses relative to n ground unit clauses has worst-case cardinality of order $O(n^m)$, making the construction of such $rlgg_B$'s intractable.

D Progol algorithm

D.1 Construction of most-specific clause

Algorithm 40 Algorithm for constructing $\perp_i$.

1. *Given natural numbers h, i, Horn clauses B, definite clause e and set of*

mode declarations M.

2. *Let $k = 0$, $hash : Terms \rightarrow N$ be a hash function which uniquely maps terms to natural numbers, $\overline{e}$ be the clause normal form logic program $\overline{a} \wedge b_1 \wedge .. \wedge b_n$, $\bot_i = \langle\rangle$ and InTerms= $\emptyset$.*

3. *If there is no modeh in M such that $a(m) \preceq a$ then return $\Box$. Otherwise let m be the first modeh declaration in M such that $a(m) \preceq a$ with substitution θ_h. Let a_h be a copy of $a(m)$ and for each v/t in θ_h if v corresponds to a #type in m then replace v in a_h by t otherwise replace v in a_h by v_k where $k = hash(t)$ and add v to InTerms if v corresponds to +type. Add a_h to $\bot_i$.*

4. *If $k = i$ return $\bot_i$ else $k = k + 1$.*

5. *For each modeb m in M let $\{v_1, .., v_n\}$ be the variables of +type in $a(m)$ and $T(m) = T_1 \times .. \times T_n$ be a set of n-tuples of terms such that each T_i corresponds to the set of all terms of the type associated with v_i in m (term t is tested to be of a particular type by calling Prolog with type(t) as goal). For each $\langle t_1, .., t_n\rangle$ in $T(m)$ let a_b be a copy of $a(m)$ and $\theta = \{v_1/t_1, .., v_n/t_n\}$. If Prolog with depth-bound h succeeds on goal $a_b\theta$ with the set of answer substitutions Θ_b then for each θ_b in Θ_b and for each v/t in θ_b if v corresponds to a #type in m then replace v in a_b by t otherwise replace v in a_b by v_k where $k = hash(t)$ and add v to InTerms if v corresponds to -type. Add $\overline{a_b}$ to $\bot_i$.*

6. *Goto step 4.*

D.2 A^*-like algorithm for finding clause with maximal compression

Firstly we define some auxiliary functions used in Algorithm 42.

Definition 41 Auxiliary functions. *Let the examples E be a set of Horn clauses. Let $h, i, B, e, M, \bot_i$ be as in Definition 24 in Section 8.1 and let C, k, θ be as in Definition 27 in Section 9.2.*

$$d'(v) = \begin{cases} 0 & \textit{if there is no -type variable in the head of } \bot_i \\ 0 & \textit{if } v \textit{ is -type in the head of } \bot_i \\ \infty & \textit{if } v \textit{ is not in } \bot_i \\ (min_{u \in U_v} d'(u)) + 1 & \textit{otherwise} \end{cases}$$

where U_v are the -type variables in atoms in the body of C which contain +type occurrences of v. Below state s has the form $\langle C, \theta, k\rangle$. c is a user-defined parameter for the maximal clause body length. $|S|$ denotes the cardinality of any set

S.

$$
\begin{aligned}
p_s &= |\{e : e \in E \ and \ B \wedge C \wedge \overline{e} \vdash_h \Box\}| \\
n_s &= |\{e : e \in E \ and \ B \wedge C \wedge e \vdash_h \Box\}| \\
c_s &= |C| - 1 \\
V_s &= \{v : u/v \in \theta \ and \ u \ in \ body \ of \ C\} \\
h_s &= min_{v \in V_s} d'(v) \\
g_s &= p_s - (c_s + h_s) \\
f_s &= g_s - n_s
\end{aligned}
$$

best(S) *is a state* $s \in S$ *which has* $c_s \leq c$ *and for which there does not exist* $s' \in S$ *for which* $f_{s'} > f_s$.

$$
\begin{aligned}
prune(s) &= \begin{cases} true & if \ n_s = 0 \ and \ f_s > 0 \\ true & if \ g_s \leq 0 \\ true & if \ c_s \geq c \\ false & otherwise \end{cases} \\
terminated(S, S') &= \begin{cases} true & if \ s = best(S), n_s = 0, f_s > 0 \ and \\ & for \ each \ s' \ in \ S' \ it \ is \ the \ case \ that \ f_s \geq g_{s'} \\ false & otherwise \end{cases}
\end{aligned}
$$

Algorithm 42 Algorithm for searching $\Box \preceq C \preceq \bot_i$.

1. *Given* $h, B, e, \bot_i$ *as in Definition 24.*
2. *Let* $Open = \{\langle \Box, \emptyset, 1 \rangle\}$ *and* $Closed = \emptyset$.
3. *Let* $s = best(Open)$ *and* $Open = Open - \{s\}$.
4. *Let* $Closed = Closed \cup \{s\}$.
5. *If* $prune(s)$ *goto 7.*
6. *Let* $Open = (Open \cup \rho(s)) - Closed$.
7. *If* $terminated(Closed, Open)$ *then return* $best(Closed)$.
8. *If* $Open = \emptyset$ *then print 'no compression' and return* $\langle e, \emptyset, 1 \rangle$.
9. *Goto 3.*

D.3 Progol's cover set algorithm

Definition 43 Unflattening. *Let $C = h \leftarrow X, Y$ be a definite clause in which $X = (s_1 = t_1, .., s_n = t_n)$ is a conjunction of atoms with predicate symbol '=' and Y is a conjunction of atoms with predicate symbols other than '='. The clause $C' = h' \leftarrow Y'$ is called the unflattening of C if and only if C' is derived from C by successively resolving away each $s_i = t_i$ in X with the clause $(U = U \leftarrow)$.*

Algorithm 44 Cover set algorithm

1. *h, i, B, M are given as in Theorem 26 and E is the subset of B corresponding to atoms in modeh declarations in M.*
2. *If $E = \emptyset$ then return B.*
3. *Let e be the first example in E.*
4. *Construct $\bot_i$ for e using Algorithm 40.*
5. *Construct state s from $\bot_i$ using Algorithm 42.*
6. *Let C' be the unflattening of $C(s)$ (Definition 43).*
7. *Let $B = B \cup C'$.*
8. *Let $E' = \{e : e \in E \text{ and } B \wedge \overline{e} \vdash_h \square\}$.*
9. *Let $E = E - E'$.*
10. *Goto 2.*

E Progol's runtimes

Data set	Predicate	$\|E^+\|$	$\|E^-\|$	$\|B\|$	$\|H\|$	Time (sec)
animals	false	42	16	105	6	0.930
	class	16	6	105	5	0.183
append	append	19	8	0	2	0.199
arch	arch	4	4	47	1	0.149
chess	move	27	12	34	11	5.080
cyclic	cyclic	3	2	69	1	0.100
delete	delete	7	6	2	2	0.365
even	even	16	15	4	3	0.216
exp	plus	6	5	13	3	0.133
	mult	6	23	10	3	0.730
	exp	5	5	8	2	0.183
family	parent_of	11	4	61	2	0.066
	grandfather_of	10	7	53	1	0.149
	grandparent_of	13	6	41	1	0.066
grammar	s	8	7	18	1	0.116
krki	illegal	341	655	51	4	17.281
last	last	7	5	2	2	0.066
min	min	14	6	4	2	1.760
nim	won	16	7	12	1	0.100
order0	f	15	3	13	1	0.382
order1	f	15	3	13	1	0.730
order2	f	8	4	13	1	0.747
order3	f	9	4	13	1	0.681
order4	f	12	4	13	1	1.079
parity4	parity	16	16	11	1	1.195
qsort	qsort	11	12	8	2	0.863
range	inrange	7	3	0	2	0.266
reverse	reverse	13	7	4	2	0.149
set	member	16	3	33	2	0.100
	pair	3	2	16	2	0.050
	subset	12	8	7	2	0.730
setuni	setuni	14	13	2	4	2.357
sumx	sumx	7	3	3	2	0.432
train	eastbound	5	5	257	1	0.100
utube	utube	5	13	173	1	1.643