Predicate Invention and Utilisation

Stephen MUGGLETON
Oxford University Computing Laboratory,
11 Keble Road,
Oxford,
OX1 3QD,
United Kingdom.
Tel: +44 865 273838
Email: steve@prg.oxford.ac.uk

Abstract Inductive Logic Programming (ILP) involves the synthesis of logic programs from examples. In terms of scientific theory formation ILP systems define observational predicates in terms of a set of theoretical predicates. However, certain basic theorems indicate that with an inadequate theoretical vocabulary this is not always possible. Predicate invention is the augmentation of a given theoretical vocabulary to allow finite axiomatisation of the observational predicates. New theoretical predicates need to be chosen from a well defined universe of such predicates. In this paper a partial order of utilisation is described over such a universe. This ordering is a special case of a logical translation. The notion of utilisation allows the definition of an equivalence relationship over new predicates. In a manner analogous to Plotkin clause refinement is defined relative to given background knowledge and a universe of new predicates. It is shown that relative least clause refinement is defined and unique whenever there exists a relative least general generalisation of a set of clauses. Results of a preliminary implementation of this approach are given.

1 Introduction

1.1 A logic programming exercise

Imagine the following. You decide to set a programming exercise as an examination question for a large class of logic-programming students. In the question you provide a logical specification of the set-union predicate. However, on receipt of the 200 student scripts you find you have a problem. Every student seems to have found a different way of doing the exercise. A variety of different data types have been used each with different data operations. You therefore decide to write a program to mark all the students' solutions. However, another problem emerges. Where one student uses the predicate 'append' which concatenates two lists, another student uses the predicate 'end' which adds an element to the end of a list. There is no way of directly comparing the models of two predicates in logic. If the predicate symbol is different or the argument ordering varies or the constant and function symbols are different two otherwise similar predicates become incomparable.

1.2 Predicate invention

The same problem turns up in the field of Inductive Logic Programming (ILP) [9, 10]. ILP involves the automatic construction of logic programs from examples and background knowledge. The theory and applications of ILP are motivated by an attempt to model two distinct areas of human activity.

1. **Program synthesis.** Construction and debugging of logic programs [18].

2. **Scientific theory formation.** The development of theories of the natural sciences from observational data and background knowledge (see [12, 4]).

In the context of logic program synthesis, ILP systems such as that of Shapiro [18] make use of the programmer to provide a complete *oracle* (extensional specification) of the input/output behaviour of each predicate within the program. In the programming exercise above this would be the equivalent of logically specifying each predicate which could allowably be used by the students.

The situation within scientific theory formation differs from Shapiro's program synthesis approach. Here the results of experimentation provide an oracle for hypothesised statements within the language of *observation*. In addition to the observation language there is a *theoretical* language of relationships which cannot be directly observed. Since no oracle exists for the theoretical language, the inductive reasoner is free to choose any convenient logical model for theoretical relationships. Thus theoretical language is, in a sense, an invented fiction (as opposed to factual observation) which is introduced to simplify the modelling of observed data. For this reason the process of introducing new theoretical relationships is called *predicate invention* within the ILP literature [13, 17, 21, 7].

Extending Shapiro's approach to program synthesis in the same way leads to a situation similar to the programming exercise problem at the beginning of this section. Namely how are invented predicates compared? A partial order over invented predicates is introduced in this paper. The semantics of this approach is a special case of that of logical translation [20]. Intuitively the partial order defines the notion of predicate utilisation. With reference to this partial order a lattice of new predicates is defined with unique elements having maximal and minimal utilisation. This notion is of central importance in defining the logical foundations of the problem of predicate invention.

This paper is organised as follows. In Section 2 the logical setting for Inductive Logic Programming is described, with special reference to the requirements for predicate invention. Results are given on finite axiomatisation, with and without predicate invention. Section 3 contains a description of the logical semantics of translation between languages with differing vocabularies. A partial order of utilisation is defined over individual predicates as a special case of a logical translation. The utilisation ordering is shown to induce a lattice over predicates in a function-free language. In Section 4 the utilisation lattice is used to define a partial order over clauses containing calls to invented predicates. The existence of a unique least upper-bound is demonstrated for such clauses. Section 5 contains results of an implementation which constructs relative least refinement clauses.

2 Inductive Logic Programming

Standard definitions from Logic Programming are given in Appendix A. The usual context for Inductive Logic Programming within this paper is as follows. The learning agent is provided with background knowledge B, positive examples E^+ and negative examples E^- and constructs an hypothesis H. B, E^+ E^- and H are each logic programs. The conditions for construction of H are

Necessity: $B \not\vdash E^+$

Sufficiency: $B \wedge H \vdash E^+$

Consistency: $B \wedge H \wedge E^- \not\vdash \square$

A logic program is a set of definite clauses each having the form

$$Head \leftarrow Body$$

Usually E^+ and E^- contain only ground clauses, with empty bodies. If P is a logic program then the set of all predicate symbols found in the heads of clauses of P is called the definitional vocabulary of P or $V(P)$. ILP has the following three definitional vocabularies.

Observational vocabulary: $\mathcal{O} = V(E^+ \cup E^-)$

Theoretical vocabulary: $\mathcal{T} = V(B) - \mathcal{O}$

Invented vocabulary: $\mathcal{I} = V(H) - (\mathcal{T} \cup \mathcal{O})$

The learner carries out predicate invention whenever $\mathcal{I} \neq \emptyset$. Without loss of generality it can be assumed that $\mathcal{I} \subseteq \mathcal{U}$ where $\mathcal{U}$ is a denumerable set of predicate symbols.

2.1 Necessary predicate invention

Ling [6] discusses the requirements for the necessary invention of predicates. This requires the following addition to the necessity, sufficiency and consistency requirements of Section 2.

Necessary invention: $\mathcal{I} \neq \emptyset$ for each H which provides sufficiency and consistency.

In other words predicate invention is only necessary when there does not exist a finite axiomatisation of the predicates in $\mathcal{O}$ containing only predicate symbols from $\mathcal{T} \cup \mathcal{O}$. The following theorem is from Stahl [19].

Theorem 1 Decidability with fixed vocabulary. *Given a recursively enumerable, deductively closed set of formulas C in a first order language $\mathcal{L}$ it is undecidable whether C is finitely axiomatisable in $\mathcal{L}$.*

Stahl's proof is based on an application of Rice's Theorem [15] on the undecidability of non-trivial index sets being recursively enumerable. This result means that the necessity of invention must by needs be heuristic in the general case. However, if constraints on the language and depth of inference such as those used in Golem [11] are applied, this problem becomes decidable.

The following result due to Kleene [5] shows the importance of the introduction of new predicates in constructing finite axiomatisations.

Theorem 2 Finite axiomatisation given additional vocabulary. *Any recursively enumerable, deductively closed set C of formulas in a first order language $\mathcal{L}$ is finitely axiomatisable using additional predicate symbols other than those in $\mathcal{L}$.*

Although Kleene's proof is constructive it introduces new predicates regardless of whether they are necessary. Clearly any one of a potentially infinite set of new predicates could be introduced. It seems reasonable that when it is necessary to extend the vocabulary this should be done in as conservative a manner as is possible. To do so requires a notion of ordering over invented predicates. This comes back to the problem of the student exercise at the start of this paper. That is the comparison of models of predicates with differing predicate symbols and argument orderings.

3 Translations and orderings

3.1 Logical translations

The problem of finding mappings between different languages is known as logical translation [3]. This approach has had applications in specifying computer programs [20] and transformational programming [1].

A translation is an interpretation which maps predicate and function symbols from the language $\mathcal{L}_1$ of theory T_1 into the language $\mathcal{L}_2$ of theory T_2 while preserving the validity of the theorems of T_1 in T_2. In particular, the translations of the axioms of T_1 must be provable from the axioms of T_2. Such an interpretation defines models of T_1 in models of T_2. The basis required to establish an interpretation of T_1 in T_2 is

- An interpretation I from $\mathcal{L}_1$ to $\mathcal{L}_2$
- Proofs of the translations of the axioms of T_1 in T_2 (including the axioms for equality if present)

The interpretation maps all predicate symbols, function symbols and constants from the language $\mathcal{L}_1$ to the language $\mathcal{L}_2$. An implementation of a theory within a language which has a different vocabulary is defined as follows.

Definition 3 *Let T_1 and T_2 be theories in first order predicate calculus. Let I be an interpretation which maps models of T_2 to models of T_1. Thus if $\mathcal{M}_2$ is a model of T_2 then $\mathcal{M}_2|I$ is a model of T_1. T_2 implements T_1 if and only if $T_2 \models_{M_2|I} T_1$.*

3.2 Utilisation ordering

Next, the ordering of utilisation over individual predicates is defined. First the definition of a canonical rewrite.

Definition 4 Canonical rewrite. *The formula $p(x_1, .., x_m) \leftrightarrow q_n$ is a canonical rewrite if and only if $x_1, .., x_m$ are distinct variables, q_n is an atom and $\{x_1, .., x_m\}$ is a subset of $vars(q_n)$.*

Lloyd [8] defines the success-set of a program.

Definition 5 Success-set. *Let P be a logic program. The success-set of P or $SS(P)$ is the set of all atoms a in the Herbrand base of P such that $P \vdash_{\text{SLD}} a$.*

The utilisation ordering $\leq_{\text{P}}$ is now defined as follows.

Definition 6 Utilisation ordering. *Let P be a logic program which does not contain the predicate symbols p^m and q^n. Let p^m_{def} be a set of clauses whose heads have the predicate symbol p^m and whose bodies contain only predicate in $V(P) \cup \{p^m\}$. Let q^n_{def} be the same for q^n. $p^m \leq_{\text{P}} q^n$ or p^m has lower utilisation than q^n in P if and only if there exists a canonical rewrite R such that $P \wedge q^n_{def} \wedge R \vdash_{\text{SLD}} SS(P \wedge p^m_{def})$.*

Note that R plays the part of a translation between the languages of $P \wedge q^m_{def}$ and $P \wedge p^n_{def}$.

Example 7 *$male^1 \leq_P son^2$ since there is a canonical rewrite $R = male(X) \leftrightarrow son(X,Y)$ such that with P empty, $son^2_{\text{def}} \wedge R \vdash SS(male^1_{\text{def}})$ and $male^1_{\text{def}} \wedge R \vdash SS(son^2_{\text{def}})$. Natural number $lessthan^2$ has lower utilisation than natural number $plus^3$ with $R = lessthan(X,Z) \leftrightarrow plus(X,Y,Z)$.*

end^3 has lower utilisation than $append^3$ with $R = end(X,Y,Z) \leftrightarrow append(X,[Y],Z)$. $harryschild^1$ has lower utilisation than $child^2$ with $R = harryschild(X) \leftrightarrow child(X, harry)$.

From these examples it can be seen that lower utilisation can be produced by changing the predicate symbol and dropping arguments, as is the case with $\text{male}^1 \leq_P \text{son}^2$ and $\text{lessthan}^2 \leq_P \text{plus}^3$. This is analogous to projecting a shape onto a lower dimensional space. The lowering of utilisation can also be viewed as partial evaluation as is the case with the relation $\text{end}^3 \leq_P \text{append}^3$. Here end^3 can be viewed as a partially instantiated call of append^3. In all cases the argument orderings and predicate symbol names are irrelevant.

3.3 Utilisation is a partial order

It is now shown that $\leq_P$ is a partial order over the set of all predicate definitions. This is done by showing that $\leq_P$ is transitive, reflexive and antisymmetric.

Lemma 8 Transitivity. *$\leq_P$ is transitive.*
Proof. *It is necessary to show that $p^m \leq_P q^n$ and $q^n \leq_P r^o$ implies $p^m \leq_P r^o$. Since $p^m \leq_P q^n$ and $q^n \leq_P r^o$ there exist canonical rewrites $p(x_1,..,x_m) \leftrightarrow q_n$ and $q(y_1,..,y_n) \leftrightarrow r_o$ where q_n and r_o are atoms with predicate symbols q^n and r^o respectively. Since the variables $\{y_1,..,y_n\}$ are distinct there must exist a substitution θ, whose domain is $\{y_1,..,y_n\}$, such that $q(y_1,..,y_n)\theta = q_n$. Thus there is a canonical rewrite $R = p(x_1,..,x_m) \leftrightarrow r_o\theta$ such that $P \wedge r^o_{def} \wedge R \vdash_{SLD} SS(P \wedge p^m_{def})$. Therefore $p^m \leq_P r^o$.* □

Lemma 9 Reflexivity. *$\leq_P$ is reflexive.*
Proof. *It is necessary to show that $p^m \leq_P p^m$. This can be shown trivially with the tautological canonical rewrite $p(x_1,..,x_m) \leftrightarrow p(x_1,..,x_m)$.* □

Utilisation equivalence is defined in terms of $\leq_P$ in the standard way.

Definition 10 Utilisation equivalence. *$p^m \equiv_P q^n$ if and only if $p^m \leq_P q^n$ and $q^n \leq_P p^m$.*

Example 11 *$child^2 \equiv_P parent^2$ using the canonical rewrite $child(X,Y) \leftrightarrow parent(X,Y)$.*

From this example it can be seen that predicate equivalence ignores the predicate symbol used and the argument ordering.

Lemma 12 Anti-symmetry. *It must be shown that* $p^m \leq_P q^n$ *and* $p^n \leq_P q^m$ *implies* $p^m \equiv_P q^n$. *This follows trivially from definition 10.*

Theorem 13 Partial order. $\leq_P$ *is a partial order.*
Proof. *From Lemmas 8, 9 and 12* $\leq_P$ *is transitive, reflexive and anti-symmetric.* □.

3.4 Utilisation as a lattice over predicates

According to Donnellan [2] a lattice can be defined in a set theoretic fashion as follows.

Definition 14 *A lattice is a partially ordered set in which every pair of elements* a, b *has a greatest lower bound (represented by* $a \cap b$*) and a least upper bound (represented by* $a \cup b$*) within the set.*

In this and following sections a restriction to function-free logic programs is made.

Theorem 15 *Let* p^m *be a predicate symbol defined in function-free logic program* P. *The set of predicates* q^n, $n \leq m$, *which can be constructed from subsets of the arguments of* p^m *form a lattice ordered by* $\leq_P$. *The minimal element in the lattice is* q^0 *and the maximal element is the predicate* p^m.
Proof. *Let* q^m *and* r^o *be formed from the arguments* $X = \{x_1, .., x_m\}$ *and* $Y = \{y_1, .., y_o\}$ *of* p^m. *The greatest lower bound* $q^m \cap r^o$ *is the predicate formed from the arguments* $X \cap Y$ *of* p^m. *The least upper bound* $q^m \cap r^o$ *is the predicate formed from the arguments* $X \cup Y$ *of* p^m. □

4 Relative clause refinement

In the last section it was shown that the set of predicates having lower utilisation than a given predicate p^m form a lattice. This section defines a partial order over clauses constructed from predicates chosen from a given universe of predicates.

Definition 16 Relative clause refinement *Let* $\mathcal{U}$ *be a denumerable set of predicate symbols. Let* $\mathcal{R}$ *be the set of canonical rewrites* $\{R : p^m \in \mathcal{U}, q^n \in V(P), p^m \leq_P q^n$ *using* $R\}$. *Let* C *and* D *be two clauses constructed using only predicate symbols from* $\mathcal{U}$. C *is finer than* D *with respect to* $\mathcal{U}$ *if and only if* $\mathcal{R} \wedge C \models D$.

Example 17 Relative clause refinement. *Let* $C = uncle(U, V) \leftarrow sibling(U, W) \wedge child(V, W) \wedge male(U)$. *Let* $D = uncle(U, V) \leftarrow sibling(U, W) \wedge child(V, W) \wedge son(U, X)$. *Let* $U = \{male^1, son^2, ...\}$. *Then* $\mathcal{R}$ *is* $\{(male(X) \leftrightarrow son(X, Y)), ...\}$ *and* $\mathcal{R} \wedge C \models D$. *Therefore* C *is finer than* D.

Note from this example that the fineness of a clause increases when higher utilisation predicates are replaced with lower utilisation ones.

Theorem 18 *Fineness partial order. Given a universe* $\mathcal{U}$ *of predicate symbols fineness forms a partial order over clauses constructed from* $\mathcal{U}$.
Proof. *Follows from the fact that relative generalisation forms a partial order [14].*

4.1 Relative least refinement

The notion of carrying maximally conservative predicate invention can now be formalised.

Theorem 19 Relative least refinement. *Let B be a logic program defining background predicates and $\mathcal{U}$ be a set of "inventable" predicates. Let C and D be two clauses. The least upper bound $C \cap_{B\cup\mathcal{U}} D$ clause in the fineness ordering is unique but may have infinite cardinality.*
Proof. *$C \cap_{B\cup\mathcal{U}} D$ is the relative least generalisation of C and D with respect to $B \cup \mathcal{U}$. Plotkin [14] shows this clause is unique though it may have infinite cardinality.* □

The question now arises as to how $\mathcal{U}$ is chosen. In this paper a simple-minded approach is taken. Suppose the relative least general generalisation of C and D with respect to B is E. Let $vars(E) = \{x_1, .., x_n\}$. Let the atom a be $p(x_1, .., x_n)$. Suppose E relatively subsumes C with substitution θ_C and E relatively subsumes D with substitution θ_D. Then $\mathcal{U}$ is the predicate lattice (see Section 3.4) whose maximal element is the predicate with definition

$a\theta_C \leftarrow$
$a\theta_D \leftarrow$

For definite clauses $C \cap_{B\cup\mathcal{U}} D$ is $rlgg(C, D, B) \vee \overline{a}$.

5 Golem implementation

The approach described in the last section has been incorporated into the ILP system Golem [11]. Since it is undecidable whether predicate invention is necessary (Theorem 1) Golem takes a heuristic approach.

1. Randomly sample pairs of instances from the positive examples E^+.
2. For each pair from the last step check whether the relative least general generalisation is consistent with E^-. If so return this clause.
3. Otherwise transform each clause from the last step to a relative least refinement by introducing a maximal utilisation predicate.
4. Construct definitions for the invented predicates from the last step.

This can produce invented predicates with a large number of arguments. However, after their definitions have been constructed redundant arguments are dropped.

The following are predicates which have been invented using this implementation.

- The predicate father-or-father-in-law^2 as an auxiliary of grandfather^2 and background knowledge about mother^2.

- The predicate plus-2^3 which defines the function (Z=X+Y-2). This was learned from example of multiply^3 and background knowledge of the successor^2.
- The predicate insert^3 from examples of list-sort^2 and background of less-than^2.

It is notable that with the first and second of these, the predicate which was invented was not as expected. However, the definitions were more compact than they would have been for the expected additional predicates. Referring back to Section 1.2 it is worth noting that in the setting of scientific theory formation the learner is free to choose any convenient model for theoretical relations.

6 Conclusion

Several systems within the ILP literature [13, 17, 21, 7] have carried out predicate invention. However, almost no theoretical results have been demonstrated. The author believes this is due, at least in part, to the lack of a theoretical framework for describing predicate invention. In this paper the logical setting for predicate invention is analysed.

A partial order over the universe of invented predicates is discussed. It is demonstrated that unique solutions exist for the most conservative addition of new predicates. This approach lends itself to efficiency since it replaces exponential search for new predicates such as that found in [13] by the construction of a unique most conservative solution. An initial implementation of this approach within Golem has been shown to give good results.

Acknowledgements.
The author would like to thank Stuart Russell and Ashwin Srinivasan for helpful and interesting discussions on the topics in this paper. This work was supported by the Esprit Basic Research Action ILP, project 6020.

Appendix

A Definitions from logic

A.1 Formulae in first order predicate calculus

A variable is represented by an upper case letter followed by a string of lower case letters and digits. A function symbol is a lower case letter followed by a string of lower case letters and digits. A predicate symbol is a lower case letter followed by a string of lower case letter and digits. The negation symbol is: $\neg$. A constant is represented as a solitary function symbol. A variable is a term, a constant is a term and a function symbol immediately followed by a bracketed n-tuple of terms is a term. Thus $f(g(X), h)$ is a term when f, g and h are function symbols and X is a variable. A predicate symbol immediately followed by a bracketted n-tuple of terms is called an atomic formula. Both A and $\neg A$ are literals whenever A is an

atomic formula. In this case A is called a positive literal and $\neg A$ is called a negative literal. The literals A and $\neg A$ are said to be each others complements and form, in either order, a complementary pair. A finite set (possibly empty) of literals is called a clause. The empty clause is represented by $\Box$. A clause represents the disjunction of its literals. Thus the clause $\{A_1, A_2, ..\neg A_i, \neg A_{i+1}, ...\}$ can be equivalently represented as $(A_1 \vee A_2 \vee ..\neg A_i \vee \neg A_{i+1} \vee ...)$ or $A_1, A_2, .. \leftarrow A_i, A_{i+1}, ...$ A Horn clause is a clause which contains at most one positive literal. The positive literal in a Horn clause is called the head of the clause while the negative literals are collectively called the body of the clause. A set of clauses is called a clausal theory. The empty clausal theory is represented by $\blacksquare$. A clausal theory represents the conjunction of its clauses. Thus the clausal theory $\{C_1, C_2, ...\}$ can be equivalently represented as $(C_1 \wedge C_2 \wedge ...)$. A set of Horn clauses is called a logic program. Apart from representing the empty clause and the empty theory, the symbols $\Box$ and $\blacksquare$ represent the logical constants *False* and *True* respectively. Literals, clauses and clausal theories are all well-formed-formulae (wff). Let E be a wff or term. vars(E) denotes the set of variables in E. E is said to be ground if and only if vars(E) $= \emptyset$.

A.2 Models and substitutions

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

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 well-formed formula 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. Every sub-term within a given term or literal W can be uniquely referenced by its *place* within W. Places within terms or literals are denoted by n-tuples of natural numbers and defined recursively as follows. The term at place $\langle i \rangle$ within $f(t_0, .., t_m)$ is t_i. The term at place $\langle i_0, .., i_n \rangle$ within $f(t_0, .., t_m)$ is the term at place $\langle i_1, .., i_n \rangle$ in t_{i_0}. Let t be a term found at place p in literal L, where L is a literal within clause C. The place of t in C is denoted by the pair $\langle L, p \rangle$. Let E be a clause or a term and $\theta = \{v_1/t_1, .., v_n/t_n\}$ be a substitution. The corresponding

inverse substitution θ^{-1} is $\{\langle t_1, \{p_{1,1}, .., p_{1,m_1}\}\rangle/v_1, .., \langle t_n, \{p_{n,1}, .., p_{1,m_n}\}\rangle/v_n\}$. An inverse substitution is applied by replacing all t_i at places $p_{i,1}, .., p_{i,m_i}$ within E by v_i. Clearly $E\theta\theta^{-1} = E$. Note that an inverse substitution is not strictly a substitution but rather a rewrite. Let C and D be clauses. C is said to θ-subsume D if and only if there exists a substitution θ such that $C\theta \subseteq D$.

A.3 Resolution

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

Let T be a clausal theory. Robinson [16] defined the function $\mathcal{R}^n(T)$ recursively as follows. $\mathcal{R}^0(T) = T$. $\mathcal{R}^n(T)$ is the union of $\mathcal{R}^{n-1}(T)$ and 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 $(\Box)$.

References

[1] Baine and Goldburg. DTRE - a semi-automatic transformation system. In B. Möller, editor, *Constructing Programs from Specifications.* North Holland, 1991.

[2] T. Donnellan. *Lattice theory.* Pergamon Press, London, 1968.

[3] H. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, 1972.

[4] 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.

[5] S. Kleene. Finite axiomatizability of theories in the predicate calculus using additional predicate symbols. In S. Kleene, editor, *Two papers on the predicate calculus, Memoirs of the American Mathematical Society No. 10.* American Mathematical Society, Providence, RI, 1952.

[6] C. Ling. Inventing necessary theoretical terms in scientific duiscovery and inductive logic programming. Technical Report 302, Dept. of Comp. Sci., Univ. of Western Ontario, 1991.

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

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

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

[10] S. Muggleton. *Inductive Logic Programming.* Academic Press, 1992.

[11] S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, London, 1992. Academic Press.

[12] S. Muggleton, R. King, and M. Sternberg. Predicting protein secondary-structure using inductive logic programming, 1992. Protein Engineering (to appear).

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

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

[15] H. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 83, 1953.

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

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

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

[19] I. Stahl. Constructive induction in inductive logic programming: an overview. Technical report, Fakultat Informatik, Universitat Stuttgart, 1992.

[20] M. Turski and T. Maibaum. *The Specification of Computer Programs.* Addison-Wesley, England, 1987.

[21] R. Wirth and P. O'Rorke. Constraints for predicate invention. In S. Muggleton, editor, *Inductive Logic Programming*, London, 1992. Academic Press.