Generalization under Implication by using Or-Introduction

Peter Idestam-Almquist

Department of Computer and Systems Sciences
Stockholm University
Electrum 230, 164 40 Kista, SWEDEN

Abstract. In the area of inductive learning, generalization is a main operation. Already in the early 1970's Plotkin described algorithms for computation of least general generalizations of clauses under θ-subsumption. However, there is a type of generalizations, called roots of clauses, that is not possible to find by generalization under θ-subsumption. This incompleteness is important, since almost all inductive learners that use clausal representation perform generalization under θ-subsumption.
In this paper a technique to eliminate this incompleteness, by reducing generalization under implication to generalization under θ-subsumption, is presented. The technique is conceptually simple and is based on an inference rule from natural deduction, called or-introduction. The technique is proved to be sound and complete, but unfortunately it suffers from complexity problems.

1 Introduction

In recent years there has been a rising interest in clausal representation of knowledge in machine learning. Generalization is a main operation for inductive algorithms, and the usual definition of induction is based on logical implication [13]. However, almost all inductive learners that use clausal representation, for example CIGOL [10], GOLEM [11], LFP2 [19], ITOU [16], perform generalizations under θ-subsumption, instead of generalization under implication. The reason is that there are well-known and reasonably efficient algorithms to compute least general generalizations under θ-subsumption [14].

The derivation rule used together with clausal representation, in logic programming and automatic theorem proving, is resolution. Resolution is refutation complete, but not derivation complete [15]. Therefore by inverting resolution [10, 8, 17, 5] we have not obtained a complete generalization procedure.

There is a type of generalizations of self-recursive clauses, called roots of clauses (see section 2.3), that is not possible to find by generalization under θ-subsumption or by inverse resolution. Consequently we need to add a new inference rule to make it possible to find such generalizations. In natural deduction there is an inference rule called *or-introduction* [18]. This rule has inspired us to develop an algorithm to reduced generalization under implication to generalization under θ-subsumption. After such a reduction the well-known algorithms for least general generalizations under θ-subsumption can be used to compute least general generalizations under implication. Our technique for reduction of generalization is proven to be sound and complete, but unfortunately it suffers from complexity problems.

Recently the problem of generalization under implication has also been studied by Muggleton [9] and Lapointe and Matwin [6]. Lapointe and Matwin describe an efficient method to find two particular types of roots of clauses. But these two types of roots only cover a fraction of all possible self-recursive clauses. Muggleton consider the general problem of inverting implication, and gives a theoretical comparison between implication and resolution. However, the algorithms he presents are restricted to a subclass of self-recursive clauses, namely single-recursive clauses.

We assume the reader to be familiar with the basic notions in logic programming [7] or automatic theorem proving [2]. In particular we shall adopt the notation in [7]. In section 2, a necessary background for our learning problem, generalization of clauses, is given. In section 3, our technique for reduction of generalization by or-introduction, is presented. Finally in section 4, related work and contributions are discussed.

2 Preliminaries

The basic definitions and terminology of generalization of clauses will be given in this section. It includes generalization under θ-subsumption, generalization under implication, and powers and roots of clauses. These notions will also be related to each other.

2.1 Generalization under θ-subsumption

As already mentioned in the introduction, least general generalization under θ-subsumption is the most common form of generalization of clauses. The following definitions are taken from [14], but the terminology is adapted to inductive logic programming.

Definition 1. A clause C *θ-subsumes* a clause C, denoted $C \preceq D$, if and only if there exists a substitution θ such that $C\theta \subseteq D$.

Example 1. The clause $C = (p(x) \leftarrow p(f(x)), p(y))$ θ-subsumes the clause $D = (p(a) \leftarrow p(f(a)), q(c))$, since $C\{x/a, y/f(a)\} \subseteq D$.

Definition 2. A clause C is a *least general generalization under θ-subsumption* (LGGθ) of a set of clauses $\{C_1, \ldots, C_n\}$, denoted $C = lgg_{\preceq}\{C_1, \ldots, C_n\}$, if and only if:

1. $C \preceq C_1, \ldots, C \preceq C_n$, and
2. for each clause D, such that $D \preceq C_1, \ldots, D \preceq C_n$, then also $D \preceq C$.

Example 2. Let $C = (p(a) \leftarrow q(a), q(f(a)))$ and $D = (p(b) \leftarrow q(f(b)))$ be clauses. Then both clauses $E = (p(x) \leftarrow q(f(x)))$ and $F = (p(x) \leftarrow q(y), q(f(x)))$ are LGGθs of C and D, which shows that LGGθs are not unique.

2.2 Generalization under Implication

We agree with Niblett [12] that there is no reason why the straight forward notion of implication, instaed of θ-subsumption, should not be used as a basis for generalization.

Definition 3. A clause C *implies* a clause D, denoted $C \Rightarrow D$, if and only if every model of C is a model of D ($C \models D$).

It is easy to see that if a clause C θ-subsumes a clause D then C implies D. But the converse does not hold which is shown by the following example.

Example 3. Let $C = (p(x) \leftarrow p(f(x)))$ and $D = (p(x) \leftarrow p(f(f(x))))$ be clauses. Then $C \Rightarrow D$, but C does not θ-subsume D.

Definition 4. A clause C is a *least general generalization under implication* (LGGI) of a set of clauses $\{C_1, \dots, C_n\}$, denoted $C = lgg_{\Rightarrow}\{C_1, \dots, C_n\}$, if and only if:

1. $C \Rightarrow C_1, \dots, C \Rightarrow C_n$, and
2. for each clause D such that $D \Rightarrow C_1, \dots, D \Rightarrow C_n$, then also $D \Rightarrow C$.

Example 4. Let $C = (p(a) \leftarrow p(f(a)), p(c))$, and $D = (p(b) \leftarrow p(f(f(b))), p(c))$ be clauses. Then for the clause $E = (p(x) \leftarrow p(f(y)), p(a))$, we have $E = lgg_{\preceq}\{C, D\}$, and for the clause $F = (p(x) \leftarrow p(f(x)), p(a))$, we have $F = lgg_{\Rightarrow}\{C, D\}$.

This example illustrates that LGGθs sometimes are over-generalizations (not least general with respect to implication), since $E \Rightarrow F$ but the converse does not hold.

2.3 Nth Powers and Nth Roots of Clauses

In [9] Muggleton introduces the notion of powers and roots of clauses, for a type of specialization and generalization of self-recursive clauses, where the clauses are resolved with themselves. The definitions of nth powers and nth roots of clauses are based on a function $\mathcal{L}$.

Definition 5. Let T be a clausal theory. Then, the *function* $\mathcal{L}$ is recursively defined as:

1. $\mathcal{L}^1(T) = T$, and
2. $\mathcal{L}^n(T) = \{R \mid C \in T, D \in \mathcal{L}^{n-1}(T)$ and R is a resolvent of C and $D\}$ $(n > 1)$.

The *resolution closure* $\mathcal{L}^*(T) = \mathcal{L}^1(T) \cup \mathcal{L}^2(T) \cup \dots$

Definition 6. A clause D is an *nth power* of a clause C if and only if D is a variant of a clause in $\mathcal{L}^n(\{C\})$ $(n \geq 1)$. We also say that C is an *nth root* of D.

Example 5. Let $C = (p(x) \leftarrow p(f(x)), p(a))$ be a clause. Then $D = (p(x) \leftarrow p(f(f(x))), p(a))$ is a second power of C, and $E = (p(x) \leftarrow p(f(f(f(x)))), p(a))$ is a third power of C. The clause $F = (p(x) \leftarrow p(f(f(x))), p(f(a)), p(a))$ is also a third power of C.

Note that E and F are not equivalent in any way. The difference between E and F is due to that the resolving literals in the resolution of C and D are not the same. Thus, this example illustrates that powers and roots of clauses are not unique.

It might be the case that a clause C implies a clause D, C does not θ-subsume D, and C is not a root of D. For this relationship we define indirect powers and indirect roots.

Definition 7. A clause D is an *indirect nth power* of a clause if and only if there exists a clause E such that $E \preceq D$ and E is an nth power of C. We also say that C is an *indirect nth root* of D.

Example 6. Let $C = (p(x) \leftarrow p(f(x)))$ and $D = (p(g(a)) \leftarrow p(f(f(g(a)))), p(b), q(c))$ be clauses. Then there exists a clause $E = (p(x) \leftarrow p(f(f(x))))$ such that E is a second power of C and $E \preceq D$. Consequently D is an indirect second power of C, and C is an indirect second root of D.

Note that a root of a clause is also, by definition, an indirect root of that clause. Important to point out is also that all generalizations of clauses under implication are indirect roots of these clauses.

3 Reduction of Generalization by Or-introduction

Our main idea is to reduce generalization under implication to generalization under θ-subsumption (reduction of generalization) by or-introduction. First we present our idea, then we infer the notion of expansion of clauses, which covers this idea. Last in this section, we prove soundness and completeness of our technique for reduction of generalization.

3.1 The Main Idea

As mentioned in the introduction, our technique for reduction of generalization makes use of the sound natural deduction rule called or-introduction. This rule says that if we have a formula E, then we can derive a new formula $E \vee F$, where F is any formula. Let D be a disjunction, then we can derive $D \vee (A \wedge \neg A)$ by or-introduction. This formula can be rewritten to $(D \vee A) \wedge (D \vee \neg A)$. Thus in clausal form we can derive a set of clauses $\{(D \cup \{A\}), (D \cup \{\neg A\})\}$, where A is an atom, from a clause D. It is of interest because we have found that if a clause C is a second root of a clause D, then there exists an atom A such that $C \preceq (D \cup \{A\})$ and $C \preceq (D \cup \{\neg A\})$. Consider the following example.

Example 7. Let $D = (p(x) \leftarrow p(f(f(x))))$ be a clause. Then $C = (p(x) \leftarrow p(f(x)))$ is a second root of D, and we have $C \Rightarrow D$ but C does not θ-subsume D. Let $A = p(f(x))$, then by or-introduction of the contradiction $(A \wedge \neg A)$ we obtain the clauses $(D \cup \{A\}) = (p(x), p(f(x)) \leftarrow p(f(f(x))))$ and $(D \cup \{\neg A\}) = (p(x) \leftarrow p(f(x)), p(f(f(x))))$. Then we have $C\{x/f(x)\} \subseteq (D \cup \{A\})$ and $C \subseteq (D \cup \{\neg A\})$. Consequently, $C \preceq (D \cup \{A\})$ and $C \preceq (D \cup \{\neg A\})$.

Our technique for reduction of generalization works just as well for nth roots as for second roots. But then we have to or-introduce $n-1$ contradictions, instead of only one. For finding a third root we or-introduce two contradictions. Let D be a clause and A and B atoms. Then by or-introduction we can get $D \vee (A \wedge \neg A) \vee (B \wedge \neg B)$, which in clausal form is a set of four clauses $\{(D \cup \{A, B\}), (D \cup \{A, \neg B\}), (D \cup \{\neg A, B\}), (D \cup \{\neg A, \neg B\})\}$. If C is a third root of D and we have chosen the right atoms A and B, then we will have $C \preceq (D \cup \{A, B\})$, $C \preceq (D \cup \{A, \neg B\})$, $C \preceq (D \cup \{\neg A, B\})$ and $C \preceq (D \cup \{\neg A, \neg B\})$. That our technique works for nth roots, and even for indirect nth roots, is indicated by the following example.

Example 8. Let $D = (p(a) \leftarrow p(f(f(f(a)))))$ be a clause. Then the clause $C = (p(x) \leftarrow p(f(x)))$ is an indirect third root of D, and we have $C \Rightarrow D$ but C does not θ-subsume D. Let $A = p(f(a))$ and $B = p(f(f(a)))$, then by or-introduction we obtain the clauses:

$$\begin{aligned}
D_1 &= (D \cup \{A, B\}) = (p(a), p(f(a)), p(f(f(a))) \leftarrow p(f(f(f(a))))),\\
D_2 &= (D \cup \{A, \neg B\}) = (p(a), p(f(a)) \leftarrow p(f(f(a))), p(f(f(f(a))))),\\
D_3 &= (D \cup \{\neg A, B\}) = (p(a), p(f(f(a))) \leftarrow p(f(a)), p(f(f(f(a))))),\\
D_4 &= (D \cup \{\neg A, \neg B\}) = (p(a) \leftarrow p(f(a)), p(f(f(a))), p(f(f(f(a))))).
\end{aligned}$$

Then we have $C\{x/f(f(a))\} \subseteq D_1$, $C\{x/f(a)\} \subseteq D_2$, $C\{x/a\} \subseteq D_3$ and $C\{x/a\} \subseteq D_4$, and thus C θ-subsumes all the clauses D_1, D_2, D_3 and D_4.

3.2 Expansion of Clauses

In the previous subsection it was illustrated how a reduction of generalization can be achieved by replacing a clause by a set of clauses. Here we will show how this set of clauses equivalently can be described by a single clause, which also is an expansion of the original clause. We start by describing our idea of or-introduction more formally.

Definition 8. Let Ω be a set of atoms, and $\{\Omega_1, \Omega_2\}$ a partition of Ω, where $\Omega_2 = \{A_1, \ldots, A_n\}$. Then $\Sigma = \Omega_1 \cup \{\neg A_1, \ldots, \neg A_n\}$ is a *sign assignment* of Ω.

Definition 9. Let D be a clause and Ω a set of atoms. Then a set of clauses *or-introduced* from D with Ω, denoted $\{D \pm \Omega\}$, is the set of clauses $\{C \cup \Sigma \mid \Sigma$ is a sign assignment of $\Omega\}$.

Example 9. Let $D = (p(a) \leftarrow p(f(f(f(a)))))$ be a clause, and $\Omega = \{p(f(a)), p(f(f(a)))\}$ a set of atoms. Then $\{D \pm \Omega\} = \{D_1, D_2, D_3, D_4\}$, where D_1, D_2, D_3 and D_4 are the same as in example 8.

By definition 2, if a clause C θ-subsumes every clause in a set of clauses, then C will also θ-subsume the LGGθ of the set of clauses. This leads us to our definition of expansion of clauses.

Definition 10. Let $\{D \pm \Omega\}$ be a set of clauses or-introduced from D with Ω. Then $E = lgg_{\preceq}\{D \pm \Omega\}$ is an *expansion* of D by Ω.

Example 10. Let D and Ω be as in example 9. Then the expansion of D by Ω is $E = lgg_{\preceq}\{D \pm \Omega\} = (p(a), p(x) \leftarrow p(f(x)), p(f(f(f(a)))))$. Hence, the third root $C = (p(x) \leftarrow p(f(x)))$ of D, which does not θ-subsume D, θ-subsumes the expansion E of D.

Expansion can be regarded as a transformation technique, since the expansion of a clause C is logically equivalent to the clause C. More important is that there always exists an expansion of a clause C such that every generalization under implication of C is reduced to a generalization under θ-subsumption. Both these results are proved in the next subsection.

Algorithm 1 Expansion of clauses

```
Input: a non-tautological clause $D$.
Output: a clause $E$ such that $E \Leftrightarrow D$, and $C \preceq E$ for every indirect root C of D ($C \Rightarrow D$).
1. Find the desirable set of atoms $\Omega$.
2. Compute the expansion $E = lgg_{\preceq}\{D \pm \Omega\}$.
```

The algorithm is non-deterministic, since we have not described how the set of atoms Ω can be found. In [4] a technique to find such a set of atoms for single-recursive clauses is described, and it is not hard to extend this technique to cover arbitrary clauses. But in this paper we are satisfied with an indeterministic version of the algorithm, since it will turn out that the second part of the algorithm is computationally intractable anyway.

3.3 Soundness and Completeness

Soundness and completeness of our expansion technique of clauses, which reduces generalization under implication to generalization under θ-subsumtion, are guaranteed by the following theorems. Theorem 12 and corollary 13 are taken from [9], and use the function $\mathcal{L}$ which is defined in section 2.3.

Theorem 11 Soundness of expansion of clauses. *Let C be a clause and Ω a set of atoms. Then $lgg_{\preceq}\{C \pm \Omega\} \Leftrightarrow C$.*

Proof. Let $E = lgg_{\preceq}\{C \pm \Omega\}$. Then for each $D \in \{C \pm \Omega\}$ we have $E \preceq D$ and thus $E \Rightarrow \{C \pm \Omega\}$. We also have $\{C \pm \Omega\} \vdash C$ (by resolution). Consequently $E \Rightarrow C$. Each literal $L_i \in C$ is included in every clause in $\{C \pm \Omega\}$, and since $lgg_{\preceq}(L_i, L_i) = L_i$ we have $C \subseteq lgg_{\preceq}\{C \pm \Omega\}$, and thus $C \preceq E$. Consequently $C \Rightarrow E$, and thus $E \Leftrightarrow C$.

Theorem 12 Subsumption theorem. *Let T be a set of clauses and C a non-tautological clause. Then $T \models C$ if and only if there exists a clause $D \in \mathcal{L}^*(T)$ such that $D \preceq C$.*

Proof. A proof can be found in [1].

Corollary 13 Implication between clauses using resolution. *Let C be a clause and D a non-tautological clause. Then $C \Rightarrow D$ ($C \models D$) if and only if there exists a clause $E \in \mathcal{L}^*(\{C\})$ such that $E \preceq D$.*

Proof. Follows directly as a special case of theorem 12.

Lemma 14 Completeness of or-introduction for roots of clauses. *Let C be a clause an D_n an nth power of C. Then there exists a set of atoms Ω_n such that $C \preceq (D_n \cup \Sigma_n)$ for each sign assignment Σ_n of Ω_n.*

Proof. The proof is by mathematical induction on n.

Base step (n=1): D_1 is a first power of C, that is $D_1 = C$. Consequently, $C \preceq (D_1 \cup \Sigma_1)$ for each sign assignment Σ_1 of any set of atoms Ω_1.

Induction hypothesis (n=k): If D_k is a kth power of C then there exists a set of atoms Ω_k such that $C \preceq (D_k \cup \Sigma_k)$ for each sign assignment Σ_k of Ω_k.

Induction step (n=k+1): Let $C = \{A\} \cup \Gamma$ and $D_k = \{B\} \cup \Lambda$, such that $A\theta_A = \overline{B}\theta_B$, where $\theta_A \cup \theta_B$ is an mgu for $\{A, \overline{B}\}$. Then a (k+1)th power of C will be $D_{k+1} = \Gamma\theta_A \cup \Lambda\theta_B$. Let $\Omega_{k+1} = \Omega_k\theta_B \cup \{A\theta_A\}$ if $A\theta_A$ is an atom, or $\Omega_{k+1} = \Omega_k\theta_B \cup \{\overline{A}\theta_A\}$ if $\overline{A}\theta_A$ is an atom. Then we distinguish between two different cases. A sign assignment Σ_{k+1} of Ω_{k+1} could be either:

1. $\Sigma_{k+1} = \Sigma_k\theta_B \cup \{A\theta_A\}$, where Σ_k is a sign assignment of Ω_k, or
2. $\Sigma_{k+1} = \Sigma_k\theta_B \cup \{B\theta_B\}$ (since $A\theta_A = \overline{B}\theta_B$), where Σ_k is a sign assignment of Ω_k.

Case 1: $D_{k+1} \cup \Sigma_{k+1} = \Gamma\theta_A \cup \Lambda\theta_B \cup \{A\theta_A\} \cup \Sigma_k\theta_B$, and thus $C\theta_A \subseteq (D_{k+1} \cup \Sigma_{k+1})$. Consequently $C \preceq (D_{k+1} \cup \Sigma_{k+1})$, which completes the proof for case 1.

Case 2: $D_{k+1} \cup \Sigma_{k+1} = \Gamma\theta_A \cup \Lambda\theta_B \cup \{B\theta_B\} \cup \Sigma_k\theta_B$, and thus $(D \cup \Sigma_k)\theta_B \subseteq (D_{k+1} \cup \Sigma_{k+1})$. Consequently $(D \cup \Sigma_k) \preceq (D_{k+1} \cup \Sigma_{k+1})$. By the induction hypothesis $C \preceq (D \cup \Sigma_k)$, and thus $C \preceq (D_{k+1} \cup \Sigma_{k+1})$ (since $\preceq$ is transitive), which completes the proof for case 2.

Theorem 15 Completeness of expansion of clauses. *Let C and D be non-tautological clauses such that $C \Rightarrow D$. Then there exists a set of atoms Ω such that $C \preceq lgg_{\preceq}\{D \pm \Omega\}$.*

Proof. By corollary 13, there exists a clause D_n such that D_n is an nth power of C and $D_n \preceq D$. Hence there exists a substitution θ such that $D_n\theta \subseteq D$. By lemma 14, there exists a set of literals Ω_n such that $C \preceq (D_n \cup \Sigma_n)$ for each sign assignment Σ_n of Ω_n. Then let $\Omega = \Omega_n\theta$, and thus $(D_n \cup \Sigma_n) \preceq (D \cup \Sigma)$ for each sign assignment Σ of Ω, where $\Sigma = \Sigma_n\theta$. Consequently $C \preceq (D \cup \Sigma)$ for each sign assignment Σ of Ω (since $\preceq$ is transitive). Then by definition 2, we have $C \preceq lgg_{\preceq}\{D \pm \Omega\}$.

4 Concluding Remarks

Almost all inductive learners that use clausal representation perform generalizations under θ-subsumption. But generalizations of a certain type, roots of clauses, are not possible to find by generalization under θ-subsumption. To eliminate this incompleteness, we have presented a technique to expand clauses so that, generalization under implication are reduced to generalization under θ-subsumption. The expansion technique has been proved to be sound and complete. It is also conceptually simple.

Recently two other approaches to the problem of generalization under implication have been presented, one by Muggleton [9] and one by Lapointe and Matwin [6]. The algorithms described in [9] are non-deterministic and restricted to single-recursive clauses. In [6] it is described how two particular types of recursive clauses, which they call purely recursive and left recursive, efficiently can be learned. However, these two types of clauses only cover a fraction of all possible self-recursive clauses.

In a recent work by Idestam-Almquist [3], which is a development of the work in [6], the class of efficiently learnable clauses is extended to cover most single-recursive clauses. By the technique he presents, the generalizations are also guaranteed to be minimally general with respect to implication.

As we have shown our expansion technique of clauses is sound and complete, which is our main contribution. Now finally we will discuss its main disadvantage, its computational complexity. The complexity of the second part of the algorithm is terrible. The or-introduced set of clauses grows exponentially with the cardinality of the set of atoms used in the or-introduction. As noted in [11] the size of an LGGθ may also grow exponentially with the number of input clauses. We recommend that an LGGθ of two clauses is reduced, by removing all redundant literals, before the next input clause is taken into account. By that the size explosion can be handled, but the time complexity problem of our technique still remains.

A question for future research is to investigate if there is a way to reduce the complexity, of the presented technique, by using some approximative and more efficient computation of LGGθs. Another direction for future research is to further extend the class of efficiently learnable clauses by the technique based on the work in [6].

Acknowledgement

This work was partially supported by NUTEK, the Swedish National Board for Technical and Industrial Development under the contract 9001734, and is a part of the work in the ESPRIT BRA ILP project 6020. The author would also like to thank Douglas Busch for inspiring discussions, and Carl-Gustaf Jansson and Fredrik Kilander for help with writing the final version of this paper.

References

1. M. Bain and S. Muggleton. Non-monotonic learning. *Machine Intelligence*, 12, 1991.
2. Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* John Wiley & Sons, 1987.
3. P. Idestam-Almquist. Generalization under implication by recursive anti-unification. Submitted to the International Workshop on Inductive Logic Programming 1993.
4. P. Idestam-Almquist. Generalization under implication. Technical report, Department of Computer and Systems Sciences, Stockholm University, 1992. Report 92-020-SYSLAB.
5. P. Idestam-Almquist. Learning missing clauses by inverse resolution. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, Ohmsha, Tokyo, 1992.
6. Stéphane Lapointe and Stan Matwin. Sub-unification: A tool for efficient induction of recursive programs. In *Proceedings of the Ninth International Conference on Machine Learning.* Morgan Kaufmann, 1992.
7. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987. Second edition.
8. Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.
9. Stephen Muggleton. Inverting implication. In Stephen Muggleton, editor, *Proceedings of the International Workshop on Inductive Logic Programming*, 1992.
10. Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning.* Morgan Kaufmann, 1988.
11. Stephen Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha Publishers.
12. Tim Niblett. A study of generalization in logic programs. In *Proceedings of the Third European Working Session on Learning.* Pitman, 1988.
13. Nilsson and Genesereth. *Logic Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.
14. G. D. Plotkin. *Automatic Methods of Inductive Inference.* PhD thesis, Edinburgh University, 1971.
15. J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1), 1965.
16. Céline Rouveirol. Extensions of inversion of resolution applied to theory completion. In Stephen Muggleton, editor, *Inductive Logic Programming.* Academic Press, San Diego, CA, 1992.
17. Céline Rouveriol and Jean François Puget. Beyond inversion of resolution. In *Proceedings of the Seventh International Conference on Machine Learning.* Morgan Kaufmann, 1990.
18. Richmond H. Thomason. *Symbolic Logic—An Introduction.* McMillan Publishers, 1970.
19. Ruediger Wirth. Completing logic programs by inverse resolution. In *Proceedings of the Fourth Working Session on Learning.* Pitman, 1989.