Generalization of Clauses Containing Cross Connections

Chotiros Surapholchai [1], *Boonserm Kijsirikul* [2], *Mark E. Hall* [3]
[1]*Institute of Computer Science, University of Heidelberg, Heidelberg, Germany*
Chotiros.Surapholchai@Informatik.Uni-Heidelberg.De
[2]*Department of Computer Engineering, Chulalongkorn University, Bangkok,Thailand*
Boonserm.K@chula.ac.th
[3]*Department of Mathematics and Computer Science, Hastings College,NE, USA*
mhall@hastings.edu

ABSTRACT

In the area of inductive learning, generalization is the main operation, and the usual definition of induction is based on logical implication. Plotkin's well-known technique for computing least general generalizations of clauses under θ-subsumption sometimes produces results which are too general with respect to implication. Muggleton has shown that this problem only occurs in one type of generalization of recursive clauses, called an indirect root. Idestam-Almquist presented a technique, called recursive anti-unification, to compute indirect roots of clauses. However there exist cases for which recursive anti-unification does not work, for example, the clauses which contain a structure called a cross connection.

In this paper, we develop a technique for computing indirect roots of Horn clauses. We first introduce a relation equivalent to the implication, called θ-proof, which is syntactically defined, using resolution and θ-subsumption. This leads to an algorithm, the *J*-algorithm, for computing indirect roots of clauses. The roots of clauses containing cross connections can be computed by the *J*-algorithm. We also prove that the output from the algorithm is a generalization under implication of the input.

KEYWORDS – Generalization, θ-subsumption, Implication, θ-proof, *J*-algorithm, Indirect root, Cross connection

1. Introduction

In the area of inductive learning, generalization is the main operation, and the usual definition of induction is based on logical implication. Least general generalizations are of special interest since they are known to be correct whenever there exist correct generalizations. In recent years there has been a rising interest in clausal representation of knowledge in machine learning. Plotkin [8] has already described a technique for computation of least general generalizations of clauses under θ-subsumption. Unfortunately, the results obtained by this technique are sometimes too general with respect to implication.

The difference between θ-subsumption and logical implication is important since almost all inductive learners that use clausal representations perform generalization under θ-subsumption rather than generalization under implication. The main reason is that there is a well known and reasonably efficient technique to compute least general generalizations under θ-subsumption, but not under implication.

Plotkin himself realized that generalization under θ-subsumption was incomplete for a certain type of clause known as a recursive clause. Recursion is an important program structure in logic programming. The ability to learn recursive clauses is therefore crucial when using a clausal representation. In [6] it is shown that the incompleteness of generalization under θ-subsumption only concerns one type of generalization of recursive clauses, which is often called an indirect root.

Idestam-Almquist [2] described the well known technique to compute least general generalizations under θ-subsumption and the most important theoretical results connected with it. He studied the theory of generalization under implication, and noted that implication between clauses is undecidable. He therefore introduced a stronger form of implication, called *T*-implication, which is decidable between clauses. He showed that for every finite set of clauses there exists a least general generalization under *T*-implication. Finally, he presented a technique, called recursive anti-unification, to compute indirect roots of clauses. Recursive anti-unification is a generalization of anti-unification, which is the usual technique for computing least general generalizations under θ-subsumption.

However, there exist cases for which recursive anti-unification does not work, cases where at least one of the clauses contains a structure called a cross connection. We will describe a technique, called the *J*-algorithm, to compute indirect roots of clauses. By this technique we can even compute roots of clauses containing cross connections, which are guaranteed to be generalizations under implication.

In this work we will only consider Horn clauses. In Section 2 background on generalization under θ-subsumption of clauses is given, and in Section 3 generalization under implication and θ-proof are discussed. In Section 4 we describe the cross connection and give our technique, the *J*-algorithm, for computing indirect roots of clauses containing the cross connection. Finally, concluding remarks are given in Section 5.

2. Generalization under θ-subsumption

In this section we will describe the framework for generalization of clauses developed by Plotkin. This framework is based on a generality relation as θ-subsumption.

Definition 1. Let C and D be clauses. Then C **θ-subsumes** D, denoted by $C \preceq D$, if and only if there exists a substitution θ such that $C\theta \subseteq D$. We also say that C is a **generalization under θ-subsumption** of D.

Proposition 2. θ-subsumption is reflexive and transitive.

Example. Consider the following clauses:
$C = (\leftarrow p(f(y)), p(x))$,
$D = (q(y) \leftarrow p(f(y)))$, and
$E = (q(a) \leftarrow p(f(a)), r(b))$.

We have $C \preceq C$ since every clause is a subset of itself. We also have $C \preceq D$ since $C\theta \subseteq D$ where $\theta = \{x/f(y)\}$, and $D \preceq E$ since $D\sigma \subseteq E$, where $\sigma = \{y/a\}$. Then $C \preceq E$, since $C\theta\sigma \subseteq E$.

Two clauses may θ-subsume each other without being variants. In other words, θ-subsumption is not anti-symmetric.

Definition 3. Let C and D be clauses. Then C and D are **equivalent under θ-subsumption**, denoted $C \sim D$, if and only if $C \preceq D$ and $D \preceq C$.

Example. Consider the following clauses:
$C = (p(a) \leftarrow q(a), q(x))$,
$D = (p(a), p(y) \leftarrow q(a))$, and
$E = (p(a) \leftarrow q(a))$.

Then we have $C \sim D$ since $C\{x/a\} \subseteq D$ and $D\{y/a\} \subseteq C$. We also have $C \sim E$ and $D \sim E$. Hence, all three clauses are equivalent under θ-subsumption. Note that no two of these clauses are variants.

We are particularly interested in least general generalizations. As already mentioned in Section 1, least general generalization under θ-subsumption is the most commonly used form of generalization of clauses.

Definition 4. A clause C is a **generalization under θ-subsumption** of a set of clauses $S = \{D_1, D_2, ..., D_n\}$ if and only if $C \preceq D_i$ for every $1 \leq i \leq n$. A generalization under θ-subsumption C of S is a **least general generalization under θ-subsumption (LGGθ)** of S if and only if $C' \preceq C$ for every generalization under θ-subsumption C' of S.

Example. Consider the following clauses:
$C = (p(b) \leftarrow q(b))$,
$D = (p(a) \leftarrow q(a), q(b))$,
$E = (p(x) \leftarrow q(x), q(b))$, and
$F = (p(x) \leftarrow q(x), q(b), q(z))$.

Both clauses E and F are LGGθs of $\{C,D\}$.

In general, an LGGθ is not unique, as shown by the example above. Plotkin showed that there exists an LGGθ of every finite set of clauses, a result which is not obvious.

3. Generalization under Implication and θ-proof

In this section we will study the theory of generalization under implication. We note that implication is difficult to work with, and therefore we will introduce a relation equivalent to implication, call θ-proof.

3.1 Implication

Implication is the most natural and straightforward basis for generalization, since the concept of inductive conclusion is defined in terms of logical consequence.

Definition 5. Let C and D be clauses. Then C **implies** D, denoted $C \Rightarrow D$, if and only if every model for C is also a model for D (i.e., $\{C\} \models D$). We also say that C is a **generalization under implication** of D.

Example. Consider the following clauses:
$C = (p(x) \leftarrow p(f(x)))$,
$D = (p(x) \leftarrow p(f^2(x)))$,
$E = (p(x) \leftarrow p(f^2(y)))$, and
$F = (p(x) \leftarrow p(f^3(x)))$.

Then we have that both C and E imply both D and F, but C does not θ-subsume D, and neither C implies E nor E implies C.

Proposition 6. Let $C_1,C_2,...,C_k,D_1,D_2,...,D_n$, and E be clauses. If $\{C_1,C_2,...,C_k\} \models D_j$ for all $j \in \{1,2,...,n\}$ and $\{D_1,D_2,...,D_n\} \models E$, then $\{C_1,C_2,...,C_k\} \models E$.

Proposition 7. Implication is reflexive and transitive.

As in the case of θ-subsumption, implication between clauses is not anti-symmetric. Also, two clauses may be equivalent under implication without being equivalent under θ-subsumption.

Definition 8. Let C and D be clauses. Then C and D are **equivalent under implication**, denoted $C \Leftrightarrow D$, if and only if $C \Rightarrow D$ and $D \Rightarrow C$.

Example. Consider the following clauses:
$C = (p(x), p(y) \leftarrow p(f(x)), p(f^2(y)))$, and
$D = (p(z) \leftarrow p(f^2(z)))$.

Then we have $C \Leftrightarrow D$. We also have that D θ-subsumes C, but C does not θ-subsume D.

The above example also shows that if a clause C implies a clause D then C does not necessary θ-subsume D. It is well known that implication is a strictly weaker relation between clauses than θ-subsumption.

Proposition 9. Let C and D be clauses. If $C \prec D$, then $C \Rightarrow D$.

Since there is no least general generalization of Horn clauses under implication, we instead turn our interest to minimally general generalizations under implication in the next definition.

Definition 10. A Horn clause C is a **minimally general generalization under implication (MinGGI)** of two Horn clauses D and E if and only if:
a) $C \Rightarrow D$ and $C \Rightarrow E$, and
b) for each Horn clause F such that $F \Rightarrow D$, $F \Rightarrow E$ and $C \Rightarrow F$, we also have $F \Rightarrow C$.

Example. Consider the following clauses:
$C = (p(a) \leftarrow p(f(a)))$,
$D = (p(b) \leftarrow p(f^2(b)))$,
$E = (p(x) \leftarrow p(f(y)))$, and
$F = (p(z) \leftarrow p(f(z)))$.

The clause E is an LGGθ of $\{C,D\}$, and F is an MinGGI of $\{C,D\}$. The MinGGI is strictly more specific than the LGGθ, since E implies F, but F does not imply E.

Example. Consider the following clauses:
$C = (p(x) \leftarrow p(f(x)))$,
$D = (p(x) \leftarrow p(f^2(x)))$,
$E = (p(x) \leftarrow p(f^2(y)))$, and
$F = (p(x) \leftarrow p(f^3(x)))$.

The both clause C and clause E are MinGGI's of D and F.

3.2 θ-proof

Definition 11. Let H be a set of clauses and B a clause. We say that H **θ-proves** B if and only if there is a sequence $A_0,A_1,A_2,...,A_n$ of clauses such that each A_i is either an element of H, or follows from A_j and A_k by resolution for some $j,k < i$, and in addition A_n θ-subsumes B. We will write $H \vdash_\theta B$ if and only if H θ-proves B. If $H = \{A\}$, we often write $A \vdash_\theta B$, and say A θ-proves B.

The following lemma is used to make it easier to prove the next theorem.

Lemma 12. Let H be a set of clauses, and let $c_1,c_2,...,c_m$ be constant symbols which do not appear in any of the clauses of H. Suppose we have a sequence $B_1,B_2,...,B_n$ of clauses such that for each i either $B_i \in H$ or there exist j, $l < i$ such that B_i follows from B_j and B_l by resolution.
Let $y_1,y_2,...,y_m$ be variables which do not occur in any of the clauses $B_1,B_2,...,B_n$. For each i, construct B_i' by replacing all occurrences of c_j with y_j, for $j = 1,2,...,m$. Then for each i either $B_i' \in H$ or there exist j, $l < i$ such that B_i' follows from B_j' and B_l' by resolution.

The following theorem shows that implication and θ-proof are equivalent.

Theorem 13. Let H be a finite set of clauses and C a clause which is not valid. Then $H \models C$ if and only if $H \vdash_\theta C$.
Proof. The proof is given in [10].

The previous theorem is similar to Theorem 4 in the paper by Nienhuys-Cheng and de Wolf [7], but the proof is different. Our proof in [10] has the advantage that in some ways it is more direct, and is done all in one case, whereas the proof in [7] uses three cases, two concerning ground clauses, and then the general case.

Collorary 14. Let $C_1,C_2,...,C_k,D_1,D_2,...,D_n$ and E be clauses. If $\{C_1,C_2,...,C_k\} \vdash_\theta D_j$ for all $j \in \{1,2,...,n\}$ and $\{D_1,D_2,...,D_n\} \vdash_\theta E$, then $\{C_1,C_2,...,C_k\} \vdash_\theta E$.

Collorary 15. θ-proof is reflexive and transitive

4. Cross Connection and the *J*-Algorithm

Idestam-Almquist's technique to compute expansions can reduce most indirect roots (see the definition below) to generalizations under θ-subsumption, but it only works when there are structural regularities called internal and/or external connections in the given clauses [2]. Such structural regularities tell how appropriate linear

expansions can be computed. However, there are proper linear indirect roots of clauses for which it is not possible to find any appropriate linear expansions by his technique. Such linear indirect roots have a kind of structure he called cross connections.

4.1 Cross Connection

Definition 16. Let $(A,\neg B)$ be an ambivalent pair of literals in a clause C, s a term in position p in A and t a term in position q in B. Then a sequence of terms $K = [s_0,s_1,...,s_n]$ is a **cross connection** with structure π from s to t if and only if:
a) $s_0 = s$ and $s_n = t$,
b) p is not a subposition of q nor is q a subposition of p, and
c) $\pi = [(p_1,q_1),(p_2,q_2),...,(p_n,q_n)]$ is a sequence of pairs of positions such that for each $0 \le i \le (n\text{-}1)$ there is a literal $L_i \in C$ such that s_i is found in position p_{i+1} in L_i and s_{i+1} is found in position q_{i+1} in L_i.

Example. Consider the following clauses:
$C = (p(x,u) \leftarrow q(x,y), r(u,v), p(v,y))$,
$D = (p(x,u) \leftarrow q(v,w), r(y,z), p(k,l))$, and
$E = (p(x,u) \leftarrow q(x,y), r(y,z), r(u,v), q(v,w), p(z,w))$.

In the clause C there is a cross connection $K_1 = [x,y]$ with structure $\pi_1 = [(<(\neg q,1)>,<(\neg q,2)>)]$ from the term x in the literal $p(x,u)$ to the term y in the literal $\neg p(v,y)$, and a cross connection $K_2 = [u,v]$ with structure $\pi_2 = [(<(\neg r,1)>,<(\neg r,2)>)]$ from the term u in the literal $p(x,u)$ to the term v in the literal $\neg p(v,y)$. The cross connections in C have disappeared in E. We have that both clauses C and D are generalizations under implication of the clause E.

Definition 17. A clause D is an $\boldsymbol{n^{th}}$ **power** of a clause C if and only if D is a variant of a clause in $L^n(\{C\})$ $(n \ge 1)$, where $L^n(\{C\})$ is the n^{th} linear resolution of {C}. We also say that C is an $\boldsymbol{n^{th}}$ **root** of D.

Definition 18. A clause D is an **indirect** $\boldsymbol{n^{th}}$ **power** of a clause C if and only if there exists a clause E such that $E \prec D$ and E is an n^{th} power of C. We also say that C is an **indirect** $\boldsymbol{n^{th}}$ **root** of D.

Example. Consider the following clauses:
$C = (p(x) \leftarrow p(f(x)))$,
$D = (p(x) \leftarrow p(f^2(x)))$,
$E = (p(x) \leftarrow p(f^3(x)))$,
$F = (p(a) \leftarrow p(f^2(a)), p(a))$, and
$G = (p(x) \leftarrow p(a))$.

The clause C is a second root of D, and a third root of E. The clause C is also an indirect second root of F, since C is a second root of D and D θ-subsumes F. The clause G is an indirect n^{th} root of itself for every $n \ge 1$. The clause G is also an indirect first root of F.

Example. Consider the following clauses:
$C = (p(x) \leftarrow p(f(x)), p(g(x)))$,
$D = (p(z) \leftarrow p(f^3(z)), p(gf^2(z)), p(gf(z)), p(g(z)))$, and
$E = (p(a) \leftarrow p(f^3(a)), p(gf^2(a)), p(gf(a)), p(g(a)))$.

The clause C is a third root of D. The clause C is also an indirect third root of E.

The following proposition shows the relation of predicate, function, and constant symbols between two clauses in which one clause implies the other. While it is not used directly in what follows, it does help to motivate the *J*-algorithm.

Proposition 19. Let A and B be Horn clauses such that B is nonvalid and $A \Rightarrow B$. Then every predicate, function, and constant symbol occurring in A must also occur in B.

There is no algorithm to compute indirect roots of some clauses that contain cross connections. Since we are interested in finding them, we create the following algorithm, called the *J*-algorithm which finds indirect roots of clauses, even if they contain cross connections.

4.2 The *J*-Algorithm

The *J*-algorithm is defined in 5 steps.

1. Input the Horn clause D.

2. Consider literals in the clause D, and create a new clause C such that C has the same positive and negative literals as clause D, but no negative literal is repeated.

3. Change the terms in the negative literals in C to new variables. Let $C_1 = C$, and $C_2 = C$.

4. Resolve the clause C_1 with C_2 to get a clause C^*. When resolving, always keep the original variables in C_1, and introduce new variables into C_2 to make the variables in C_2 disjoint from those in C_1. Also, when unifying, never replace a variable in C_1 with one of the new variables introduced into C_2.

5. Choose one literal which occurs only as a negative literal, and let n be the number of times that literal occurs in C^*, and m the number of times it occurs in D.

If $n = m$, then
  if we can obtain D from C^* by substituting for some variables which do not occur in C,
    then finish, with the output C,
  else find a substitution θ such that $C^*\theta = D$, replace C with $C\theta$,
    let $C_1 = C$, $C_2 = C$, and go back to step 4,
else let $C_1 = C$ and $C_2 = C^*$, and go back to step 4.

Proposition 20. Let D be a Horn clause and C the Horn clause which is the output from the *J*-algorithm when the input is D. Then C is a generalization under implication of clause D and an indirect root of D.

The following example shows how the *J*-algorithm can find an indirect root of a clause that contains a cross connection.

Example. Let $D = (p(x,y) \leftarrow q(x,f(z)), q(y,f(w)), p(f(z), f(w)))$, and note that D contains a cross connection. Let us find an indirect root C of D using the *J*-algorithm.

We start with $C = (p(x,y) \leftarrow q(m,n), p(k,l))$, and $C_1 = C_2 = C$.
- Resolve C_1 with C_2,
  $C_1 = (p(x,y) \leftarrow q(m,n), p(k,l))$, and
  $C_2 = (p(x',y') \leftarrow q(m',n'), p(k',l'))$.
- Apply $\{x'/k, y'/l\}$ to C_2,
  $C_1 = (p(x,y) \leftarrow q(m,n), p(k,l))$, and
  $C_2 = (p(k,l) \leftarrow q(m',n'), p(k',l'))$.
- Resolve on $p(k,l)$, we get
  $C_s = (p(x,y) \leftarrow q(m,n), q(m',n'), p(k',l'))$.
- There are two ways to substitute for m and n that will allow C_s to match D: $\{m/x, n/f(z)\}$, or $\{m/y, n/f(w)\}$.

Case I. $\{m/x, n/f(z)\}$. Then let $C = (p(x,y) \leftarrow q(x,f(z)), p(k,l))$, and $C_1 = C_2 = C$.
- Resolve C_1 with C_2,
  $C_1 = (p(x,y) \leftarrow q(x,f(z)), p(k,l))$, and
  $C_2 = (p(x',y') \leftarrow q(x',f(z')), p(k',l'))$.
- Apply $\{x'/k, y'/l\}$ to C_2,
  $C_1 = (p(x,y) \leftarrow q(x,f(z)), p(k,l))$, and
  $C_2 = (p(k,l) \leftarrow q(k,f(z')), p(k',l'))$.
- Resolve on $p(k,l)$, we get
  $C_s = (p(x,y) \leftarrow q(x,f(z)), q(k,f(z')), p(k',l'))$.
- To allow C_s to match D we must make the substitution $\{k/y\}$.

So, let $C = (p(x,y) \leftarrow q(x,f(z)), p(y,l))$, and $C_1 = C_2 = C$.
- Resolve C_1 with C_2,

$C_1 = (p(x,y) \leftarrow q(x,f(z)), p(y,l))$, and
$C_2 = (p(x',y') \leftarrow q(x',f(z')), p(y',l'))$.

- Apply $\{x'/y, y'/l\}$ to C_2,
  $C_1 = (p(x,y) \leftarrow q(x,f(z)), p(y,l))$, and
  $C_2 = (p(y,l) \leftarrow q(y,f(z')), p(l,l'))$.
- Resolve on $p(y,l)$, we get
  $C_s = (p(x,y) \leftarrow q(x,f(z)), q(y,f(z')), p(l,l'))$.
- This shows us we need to make the substitution $\{l/f(z)\}$.

Hence, let $C = (p(x,y) \leftarrow q(x,f(z)), p(y, f(z)))$, and $C_1 = C_2 = C$.

- Resolve C_1 with C_2,
  $C_1 = (p(x,y) \leftarrow q(x,f(z)), p(y, f(z)))$, and
  $C_2 = (p(x',y') \leftarrow q(x',f(z')), p(y', f(z')))$.
- Apply $\{x'/y, y'/f(z)\}$ to C_2,
  $C_1 = (p(x,y) \leftarrow q(x,f(z)), p(y, f(z)))$, and
  $C_2 = (p(y,f(z)) \leftarrow q(y,f(z')), p(,f(z), f(z')))$.
- Resolve on $p(y, f(z))$, we get
  $C_s = (p(x,y) \leftarrow q(x,f(z)), q(y,f(z')), p(,f(z), f(z')))$.
- The substitution $\{z'/w\}$ makes C_s match D. But z' is not a variable in C, so in this case we are finished, with $C = (p(x,y) \leftarrow q(x,f(z)), p(y,f(z)))$.

Case II. $\{m/y, n/f(w)\}$. Then let $C = (p(x,y) \leftarrow q(y,f(w)), p(k,l))$, and $C_1 = C_2 = C$.

- Resolve C_1 with C_2,
  $C_1 = (p(x,y) \leftarrow q(y,f(w)), p(k,l))$, and
  $C_2 = (p(x',y') \leftarrow q(y',f(w')), p(k',l'))$.
- Apply $\{x'/k, y'/l\}$ to C_2,
  $C_1 = (p(x,y) \leftarrow q(y,f(w)), p(k,l))$, and
  $C_2 = (p(k,l) \leftarrow q(l,f(w')), p(k',l'))$.
- Resolve on $p(k,l)$, we get
  $C_s = (p(x,y) \leftarrow q(y,f(w)), q(l,f(w')), p(k',l'))$.
- This shows we need to make the substitution $\{l/x\}$.

So, let $C = (p(x,y) \leftarrow q(y,f(w)), p(k,x))$, and $C_1 = C_2 = C$.

- Resolve C_1 with C_2,
  $C_1 = (p(x,y) \leftarrow q(y,f(w)), p(k,x))$, and
  $C_2 = (p(x',y') \leftarrow q(y',f(w')), p(k',x'))$.
- Apply $\{x'/k, y'/x\}$ to C_2,
  $C_1 = (p(x,y) \leftarrow q(y,f(w)), p(k,x))$, and
  $C_2 = (p(k,x) \leftarrow q(x,f(z')), p(k',k))$.
- Resolve on $p(k,x)$, we get
  $C_s = (p(x,y) \leftarrow q(y,f(w)), q(x,f(z')), p(k',k))$.
- Now we need the substitution $\{k/f(w)\}$.

Hence, let $C = (p(x,y) \leftarrow q(y,f(w)), p(f(w),x))$, and $C_1 = C_2 = C$.

- Resolve C_1 with C_2,
  $C_1 = (p(x,y) \leftarrow q(y,f(w)), p(f(w),x))$, and
  $C_2 = (p(x',y') \leftarrow q(y',f(w')), p(f(w'),x'))$.
- Apply $\{x'/f(w), y'/x\}$ to C_2,
  $C_1 = (p(x,y) \leftarrow q(y,f(w)), p(f(w),x))$, and
  $C_2 = (p(f(w),x) \leftarrow q(x,f(w')), p(f(w'),f(w)))$.
- Resolve on $p(f(w),x)$, we get
  $C_s = (p(x,y) \leftarrow q(y,f(w)), q(x,f(w')), p(f(w'),f(w)))$.
- Now the substitution $\{w'/z\}$ makes C_s equal D. Again, w' does not appear in C, so we are finished, with $C = (p(x,y) \leftarrow q(y,f(w)), p(f(w),x))$ this time.

Consequently, from $D = (p(x,y) \leftarrow q(x,f(z)), q(y,f(w)), p(f(z),f(w)))$, we get two indirect roots from the J-algorithm,

$C_1 = (p(x,y) \leftarrow q(x,f(z)), p(y,f(z)))$, and
$C_2 = (p(x,y) \leftarrow q(y,f(w)), p(f(w),x))$.
Note that C_1 and C_2 themselves contain cross connections.

5. Concluding Remarks

We have presented an algorithm, the *J*-algorithm, for computing indirect roots of Horn clauses. With the *J*-algorithm we can compute roots of some clauses whose roots could not be computed before, because they contained cross connections. However, it is not clear that the *J*-algorithm can find indirect roots of all clauses. As a result, further study is required, which may lead to an enhanced algorithm.

References

[1] Bratko, I., *Prolog Programming for Artificial Intelligence*, Addison Wesley, 1990.
[2] Idestam-Almquist, P., *Generalization of Clause*, PhD thesis, Stockholm University and the Royal Institute of Technology, Sweden, 1993.
[3] Idestam-Almquist, P., "Generalization under Implication by Recursive Anti-unification", *International Conference on Machine Learning*, Vol. 10, 1993. pp.151-158
[4] Le, T. V. , *Techniques of Prolog Programming*, John Wiley and Sons Inc., 1993.
[5] Lloyd, J. W. , *Foundations of Logic Programming*, Springer-Verlag, 1984.
[6] Muggleton, S., "Inverting Implication", *Artificial Intelligence Journal*, 1993.
[7] Nienhuys-Cheng, S.-H. and de Wolf, R. M., "The Subsumption Theorem in Inductive Logic Programming : Facts and Fallacies", *Workshop on Inductive Logic Programming* (ILP 95), Germany, 1995.
[8] Plotkin, G. D., *Automatic Methods of Inductive Inference*, PhD thesis, Edinburgh University, UK, 1971.
[9] Robinson, J. A., "A Machine-oriented Logic based on the Resolution Principle", *Journal of ACM*, Vol. 12, No. 1, 1965. pp.23-41
[10] Surapholchai, C., *Generalization of Certain Types of Clauses of the First-order Predicate Logic*, Master's Thesis, Chulalongkorn University, Thailand, 1996.