Some Properties of Inverse Resolution in Normal Logic Programs

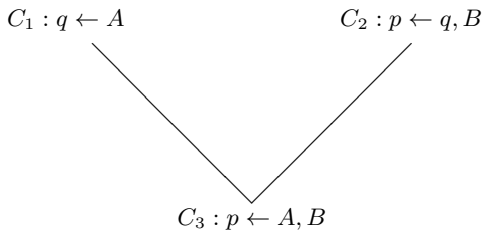
Chiaki Sakama

Department of Computer and Communication Sciences
Wakayama University
Sakaedani, Wakayama 640 8510, Japan
sakama@sys.wakayama-u.ac.jp
http://www.sys.wakayama-u.ac.jp/~sakama

Abstract. This paper studies the properties of inverse resolution in normal logic programs. The V-operators are known as operations for inductive generalization in definite logic programs. In the presence of negation as failure in a program, however, the V-operators do not work as generalization operations in general and often make a consistent program inconsistent. Moreover, they may destroy the syntactic structure of logic programs such as acyclicity and local stratification. On the procedural side, unrestricted application of the V-operators may lose answers computed in the original program and make queries flounder. We provide sufficient conditions for the V-operators to avoid these problems.

1 Introduction

Inverse resolution introduced in [13] is known as operations which perform inductive generalization in definite logic programs. There are two operators that carry out inverse resolution, *absorption* and *identification*, which are called the *V-operators* together. Each operator builds one of the two parent clauses given the other parent clause and the resolvent. More precisely, absorption constructs C_2 from C_1 and C_3, while identification constructs C_1 from C_2 and C_3 in the figure (where lower-case letters are atoms and upper-case letters are conjunction of atoms).



Absorption and identification are realized in Duce [10, 11] for propositional Horn theories, and a restricted version of absorption is implemented in CIGOL [13] for predicate Horn theories.

The V-operators are considered as program transformation rules. That is, absorption transforms the set of clauses $\{ C_1, C_3 \}$ to $\{ C_1, C_2 \}$, while identification transforms $\{ C_2, C_3 \}$ to $\{ C_1, C_2 \}$. In logic programming, program transformation is an important technique for program development, especially for deriving an efficient program preserving the meaning of the original program. In the context of *inductive logic programming* (ILP), program transformation is used for a different purpose. In ILP the original program represents an imperfect background theory, and it is transformed to a more general/specific program which covers given positive/negative evidences. The V-operators are used as program transformation rules which generalize definite Horn logic programs.

When a program is *nonmonotonic*, however, the behavior of the V-operators is not clear. The importance of nonmonotonic reasoning in commonsense inference is widely recognized, and many studies have been done to formalize nonmonotonic reasoning in logic programming [6]. In logic programming, nonmonotonic reasoning is realized using *negation as failure*, and a program with negation as failure is called a *normal logic program*. Then, our primary interest is the semantic nature of the V-operators in normal logic programs.

This paper investigates the properties of inverse resolution in normal logic programs. In the presence of negation as failure, we show that the V-operators do not work as generalization operators in general and often make a consistent program inconsistent. Moreover, the V-operators destroy the structures of logic programs such as acyclicity and local stratification. On the procedural side, it is shown that unrestricted application of the V-operators may lose answers computed in the original program and make queries flounder. We provide sufficient conditions for the V-operators to avoid these problems.

The paper is organized as follows. Section 2 reviews the logic programming framework considered in this paper. Section 3 shows the declarative properties of inverse resolution in normal logic programs. Section 4 argues the effects of the V-operators on query-answering. Section 5 presents related issues and Section 6 concludes the paper.

2 Normal Logic Programs

A *normal logic program* is a set of rules of the form:

$$p \leftarrow q_1, \ldots, q_m, \neg q_{m+1}, \ldots, \neg q_n \tag{1}$$

where p and q_i $(1 \leq i \leq n)$ are atoms and $\neg$ presents *negation as failure*. Throughout the paper a program means a normal logic program unless stated otherwise. The left-hand side of the rule is the *head*, and the right-hand side is the *body*. A rule with an empty body is a *fact*. A program P is *definite* if no rule in P contains negation as failure. A program, a rule or an atom is *ground* if it contains no variable. A program P is semantically identified with its ground instantiation, i.e., the set of ground rules obtained from P by substituting variables in P by elements of its Herbrand universe in every possible way.

A partial order relation $\geq$ is defined over the Herbrand base HB of a program such that: if $p \geq q$ then p is in a level higher than or equal to q. A program P is *locally stratified* [15] if (i) for any ground rule of the form (1) from P, $p \geq q_i$ $(i = 1, \ldots, m)$ and $p > q_j$ $(j = m+1, \ldots, n)$; and (ii) there is no infinite sequence such as $p_1 > p_2 > \cdots$. Here, $p > q$ if $p \geq q$ and $q \ngeq p$. A program P is *acyclic* [2] if (i) for any ground rule of the form (1) from P, $p > q_i$ $(i = 1, \ldots, n)$; and (ii) the same condition as above. By definition, the class of acyclic programs is strictly included in the class of locally stratified programs.

An interpretation I $(\subseteq HB)$ *satisfies* the conjunction $C = q_1, \ldots, q_m, \neg q_{m+1}, \ldots, \neg q_n$ if $\{q_1, \ldots, q_m\} \subseteq I$ and $\{q_{m+1}, \ldots, q_n\} \cap I = \emptyset$ (written as $I \models C$). I satisfies the rule (1) if $\{q_1, \ldots, q_m\} \subseteq I$ and $\{q_{m+1}, \ldots, q_n\} \cap I = \emptyset$ imply $p \in I$. An interpretation I which satisfies every rule in a program P is a *model* of the program (written as $I \models P$). A model I of a program P is called *supported* [1] if for each atom p in I, there is a ground rule (1) from P such that I satisfies its body. A model I of P is *minimal* if there is no model J of P such that $J \subset I$. A definite program P has the unique minimal model which is the *least model* (denoted by LM_P).

For the semantics of normal logic programs, we consider the *stable model semantics* of [8] and Clark's *completion* [7]. Given a program P and an interpretation I, the ground definite program P^I is defined as follows: a ground rule $p \leftarrow q_1, \ldots, q_m$ is in P^I iff there is a ground rule of the form (1) in the ground instantiation of P such that $\{q_{m+1}, \ldots, q_n\} \cap I = \emptyset$. If the least model of P^I is identical to I, I is called a *stable model* of P. A program may have none, one, or multiple stable models in general. In a definite program, a stable model coincides with the least model. A locally stratified program has the unique stable model which is called the *perfect model*. A program is *consistent* (under the stable model semantics) if it has a stable model; otherwise it is *inconsistent*. For a consistent program P and an atom a, if a is included in every stable model of P, it is written as $P \models a$; otherwise $P \not\models a$.

On the other hand, suppose that a ground program P has k rules $p \leftarrow B_1; \ldots; p \leftarrow B_k$ defining the predicate p. Then, the completion of a program P (written as $comp(P)$) includes the first-order formula $p \leftrightarrow B_1 \vee \cdots \vee B_k$.[1] In particular, when P has no definition of p, $p \leftrightarrow false$ is in $comp(P)$. We say that an interpretation I is a *completion model* of P if I is a minimal model of $comp(P)$. The (in)consistency of a program under the completion semantics is defined in the same way as the stable model semantics.

3 Declarative Properties

This section investigates the declarative properties of inverse resolution, and programs and rules are assumed to be ground.

[1] In this paper we consider the completion of a ground program. $\neg$ is interpreted as classical negation in $comp(P)$.

3.1 Semantic Properties

Absorption and *identification* are operations such that

Absorption:

$$\begin{aligned} Input:\ & C_1 : q \leftarrow A \text{ and } C_3 : p \leftarrow A, B \\ Output:\ & C_2 : p \leftarrow q, B \ \text{ and } \ C_1 \end{aligned} \tag{2}$$

Identification:

$$\begin{aligned} Input:\ & C_2 : p \leftarrow q, B \text{ and } C_3 : p \leftarrow A, B \\ Output:\ & C_1 : q \leftarrow A \ \text{ and } \ C_2 \end{aligned} \tag{3}$$

where p and q are atoms, and A and B are conjunctions in the body. Throughout the paper, we use the symbols C_1, C_2, and C_3 which refer to rules of the above forms. Absorption and identification are called the *V-operators* together. Note that in [13] these operations are introduced to definite programs. Here we consider the V-operators in normal logic programs.

In the ILP literature, there are two cases in the usage of these operations. The first case is that the input rule C_1 in absorption or C_2 in identification is included in the background theory P, while the input rule C_3 is given as an example aside from P (e.g. [13, 16, 17]). In this case, the output rule C_2 in (2) or C_1 in (3) is just added to the original theory P. The second case is that the input rule C_3 is also included in the background theory P as well as the input rule C_1 or C_2 (e.g. [5, 10, 11]). In this case, the input rule C_3 in P is replaced by the output rule C_2 in (2) or C_1 in (3).[2]

In this section, the distinction of two cases is not important. In fact, the properties presented in Sections 3.1 and 3.2 hold in either case. Then, we do not distinguish the locations of the input rules and consider the second case in this section.[3] Given a program P containing the rules C_1 and C_3, absorption produces the program $A(P)$ such that

$$A(P) = (P \setminus \{C_3\}) \cup \{C_2\}\,.$$

On the other hand, given a program P containing the rules C_2 and C_3, identification produces the program $I(P)$ such that

$$I(P) = (P \setminus \{C_3\}) \cup \{C_1\}\,.$$

Note that there are multiple $A(P)$ or $I(P)$ exist in general according to the choice of the input rules in P.

For notational convenience, we use $V(P)$ which means either $A(P)$ or $I(P)$. With this setting, absorption (resp. identification) is captured as a program transformation from P to $A(P)$ (resp. $I(P)$). Then, we first investigate semantical relations between the original program P and the produced program $V(P)$.

[2] C_3 is derived from C_1 and C_2, hence it is redundant.

[3] The distinction between two cases makes sense in Section 4. We will notice this point in Section 4.

Proposition 3.1. *Let P be a normal logic program. If M is a stable model of $V(P)$, M is a model of P.*

Proof. When M is a stable model of $V(P)$, M satisfies C_1 and C_2 in $V(P)$. If $M \not\models A$, M satisfies C_3. Else if $M \models A$, $q \in M$ by C_1. As M satisfies C_2, $M \models B$ implies $p \in M$. Hence, M satisfies C_3. Therefore, M is a model of P. □

Corollary 3.2 *Let P be a definite program and $LM_{V(P)}$ a least model of $V(P)$. Then, $LM_P \subseteq LM_{V(P)} \subseteq HB$.*

Proof. Since $LM_{V(P)}$ is a model of P by Proposition 3.1, the result follows immediately. □

In the above corollary, when $LM_{V(P)} \backslash LM_P \neq \emptyset$, any atom $p \in LM_{V(P)} \backslash LM_P$ is often called an *inductive leap* in the literature.

Proposition 3.1 presents that any stable model M of the produced program $V(P)$ satisfies the original program P (i.e., $M \models P$). Clearly, a stable model M of $V(P)$ is not necessarily a stable model of P. Indeed, M is neither minimal nor supported in P in general.

Proposition 3.3. *A stable model M of $V(P)$ is generally neither minimal nor supported in P.*

Example 3.1. Let P_1 be the program

$$p \leftarrow q, \quad r \leftarrow q, \quad r \leftarrow .$$

Using the first rule and the second rule, absorption produces $A(P_1)$:

$$p \leftarrow r, \quad r \leftarrow q, \quad r \leftarrow .$$

Here, $A(P_1)$ has the stable model $\{ p, r \}$, which it is neither minimal nor supported in P_1. Next, let P_2 be the program

$$p \leftarrow q, r, \quad p \leftarrow s, r, \quad s \leftarrow .$$

Using the first rule and the second rule, identification produces $I(P_2)$:

$$p \leftarrow q, r, \quad q \leftarrow s, \quad s \leftarrow .$$

Here, $I(P_2)$ has the stable model $\{ q, s \}$, which is neither minimal nor supported in P_2.

The V-operators may increase proven facts hence a stable model of $V(P)$ is not a minimal model of P in general. On the other hand, absorption generalizes the condition of a rule, so that the body of the original rule may not be requested to be true for implying the head of the rule in $A(P)$. Also, identification may produce a rule having the head with an atom that does not appear in the head of any rule in the original program. When a stable model of $I(P)$ contains such an atom, it may not be supported in P.

Next we consider the completion semantics. It is shown that a completion model of $V(P)$ is a model of P.

Proposition 3.4. *Let P be a normal logic program. When M is a completion model of $V(P)$, M is a model of P.*

Proof. Suppose that C_1 and C_2 are respectively completed as $C_1' : q \leftrightarrow A \vee \Gamma_1$ and $C_2' : p \leftrightarrow q, B \vee \Gamma_2$ in $V(P)$, where Γ_1 and Γ_2 are formulas in disjunctive normal forms. When M is a completion model of $V(P)$, M satisfies C_1' and C_2'. Then, M satisfies $p \leftrightarrow ((A \vee \Gamma_1) \wedge B) \vee \Gamma_2 = p \leftrightarrow (A \wedge B) \vee (\Gamma_1 \wedge B) \vee \Gamma_2$. Hence, M satisfies $C_3 : p \leftarrow A, B$. Therefore, M is a model of P. □

A completion model of $V(P)$ is not necessarily a completion model of P, and vice versa.

Example 3.2. Let P be the program

$$p \leftarrow r, s, \quad q \leftarrow r, \quad q \leftarrow t, \quad s \leftarrow, \quad t \leftarrow .$$

Then $comp(P)$ becomes

$$p \leftrightarrow r, s, \quad q \leftrightarrow r \vee t, \quad r \leftrightarrow false, \quad s \leftrightarrow true, \quad t \leftrightarrow true,$$

which has the completion model $\{q, s, t\}$. On the other hand, using the first rule and the second rule in P, absorption produces $A(P)$:

$$p \leftarrow q, s, \quad q \leftarrow r, \quad q \leftarrow t, \quad s \leftarrow, \quad t \leftarrow .$$

Then $comp(V(P))$ becomes

$$p \leftrightarrow q, s, \quad q \leftrightarrow r \vee t, \quad r \leftrightarrow false, \quad s \leftrightarrow true, \quad t \leftrightarrow true,$$

which has the completion model $\{p, q, s, t\}$.

From a consistent program P, the V-operators may produce an inconsistent program $V(P)$.

Example 3.3. Let P_1 be the program

$$p \leftarrow q, \neg p, \quad q \leftarrow r, \quad s \leftarrow r, \quad s \leftarrow,$$

which has the stable model $\{s\}$. Using the second rule and the third rule, absorption produces $A(P_1)$:

$$p \leftarrow q, \neg p, \quad q \leftarrow s, \quad s \leftarrow r, \quad s \leftarrow,$$

which has no stable model. Next, let P_2 be the program

$$p \leftarrow q, \neg p, \quad r \leftarrow q, \quad r \leftarrow s, \quad s \leftarrow,$$

which has the stable model $\{r, s\}$. Using the second rule and the third rule, identification produces $I(P_2)$:

$$p \leftarrow q, \neg p, \quad r \leftarrow q, \quad q \leftarrow s, \quad s \leftarrow,$$

which has no stable model.

The same problem happens under the completion semantics, e.g.,

$$comp(P_1) = \{\, p \leftrightarrow q,\, \neg p, \quad q \leftrightarrow r, \quad r \leftrightarrow false, \quad s \leftrightarrow r \vee true \,\}$$

is consistent, while

$$comp(A(P_1)) = \{\, p \leftrightarrow q,\, \neg p, \quad q \leftrightarrow s, \quad r \leftrightarrow false, \quad s \leftrightarrow r \vee true \,\}$$

is inconsistent. Therefore, it is concluded that:

Proposition 3.5. *The V-operators may turn a consistent normal logic program into inconsistent under both the stable model semantics and the completion semantics.*

This problem does not arise in definite programs since a definite program is always consistent. A sufficient condition for guaranteeing the consistency of the produced program $V(P)$ is given in the next section.

Next we consider the use of the V-operators as generalization operators. We say that a program P_1 *generalizes* a program P_2 if $P_2 \models a$ implies $P_1 \models a$ for any atom a. The V-operators generalize a program P when P is definite, but this is not the case in the presence of negation as failure in general.

Example 3.4. Let P_1 be the program

$$p \leftarrow \neg q, \quad q \leftarrow r, \quad s \leftarrow r, \quad s \leftarrow,$$

which has the stable model $\{p, s\}$. Using the second rule and the third rule, absorption produces $A(P_1)$:

$$p \leftarrow \neg q, \quad q \leftarrow s, \quad s \leftarrow r, \quad s \leftarrow,$$

which has the stable model $\{q, s\}$. Then, $P_1 \models p$ but $A(P_1) \not\models p$.

Next, let P_2 be the program

$$p \leftarrow \neg r, \quad q \leftarrow r, \quad q \leftarrow s, \quad s \leftarrow,$$

which has the stable model $\{p, q, s\}$. Using the second rule and the third rule, identification produces $I(P_2)$:

$$p \leftarrow \neg r, \quad q \leftarrow r, \quad r \leftarrow s, \quad s \leftarrow,$$

which has the stable model $\{q, r, s\}$. Then, $P_2 \models p$ but $I(P_2) \not\models p$.

The same phenomenon is observed under the completion semantics. In nonmonotonic theories, newly proven facts may block the derivation of other facts which are proven beforehand. As a result, the V-operators may not generalize the original program. Note that the above two programs P_1 and P_2 are (locally) stratified, which are the simplest extension of definite programs.

Proposition 3.6. *The V-operators do not generalize a normal logic program in general. This is the case even if a program is locally stratified.*

A simple condition for absorption (resp. identification) to generalize a normal logic program P is that for any negative literal $\neg a$ in P, a does not depend on p of C_3 (resp. q of C_2) in P.[4]

3.2 Syntactic Properties

In normal logic programs, syntactic restrictions on a program often introduce some nice properties in both the declarative and the procedural aspects. For instance, a locally stratified program always has a unique stable model, and an acyclic program guarantees termination of a top-down proof procedure. Therefore, when considering any program transformation, the preservation of such syntactic structures is particularly important to keep those nice properties in the transformed program. Unfortunately, the V-operators may destroy the structure of both acyclicity and local stratification.

Example 3.5. Let P_1 be the locally stratified (and also acyclic) program

$$p \leftarrow q, \quad r \leftarrow q, \quad r \leftarrow \neg\, p,$$

where $p \geq q$, $r \geq q$, $r > p$. Using the first rule and the second rule, absorption produces $A(P_1)$:

$$p \leftarrow r, \quad r \leftarrow q, \quad r \leftarrow \neg\, p,$$

where $p \geq r$, $r \geq q$, $r > p$. Here $p \geq r$ conflicts with $r > p$, hence $A(P_1)$ is neither acyclic nor locally stratified.

Next, let P_2 be the locally stratified (and also acyclic) program

$$p \leftarrow q, r, \quad p \leftarrow q, \neg\, s, \quad s \leftarrow \neg\, r,$$

where $p \geq q$, $p \geq r$, $p > s$, $s > r$. Using the first rule and the second rule, identification produces $I(P_2)$:

$$p \leftarrow q, r, \quad r \leftarrow \neg\, s, \quad s \leftarrow \neg\, r,$$

where $p \geq q$, $p \geq r$, $r > s$, $s > r$. Here $r > s$ conflicts with $s > r$, hence $I(P_2)$ is neither acyclic nor locally stratified.

Proposition 3.7. *Given a locally stratified (resp. acyclic) program P, $V(P)$ is not locally stratified (resp. acyclic) in general.*

We give a sufficient condition for the V-operators to preserve such syntactic structures of the original program.

Proposition 3.8. *Let P be a locally stratified (resp. acyclic) program.*

(i) Suppose that the rule C_2 is produced from C_1 and C_3 by absorption of (2). If the relation $p \geq q$ (resp. $p > q$) holds in P, $A(P)$ is also locally stratified (resp. acyclic).

[4] Here, *depends on* is a transitive relation defined as: p depends on q if there is a ground rule from P s.t. p appears in the head and q appears in the body of the rule.

(ii) Suppose that the rule C_1 is produced from C_2 and C_3 by identification of (3). For any positive literal a_i in A and any negative literal $\neg a_j$ in A, if the relations $q \geq a_i$ (resp. $q > a_i$) and $q > a_j$ hold in P, $I(P)$ is also locally stratified (resp. acyclic).

Proof. (i) When P is locally stratified (resp. acyclic), for any positive literal b_i in B of C_3 and any negative literal $\neg b_j$ in B of C_3, the relations $p \geq b_i$ (resp. $p > b_i$) and $p > b_j$ hold. In addition, the relation $p \geq q$ (resp. $p > q$) holds in P by assumption, then the rule C_2 produced by absorption satisfies the condition of local stratification (resp. acyclicity).

(ii) Since the relations $q \geq a_i$ (resp. $q > a_i$) and $q > a_j$ hold in P, the rule C_1 produced by identification satisfies the condition of local stratification (resp. acyclicity). Hence, the result follows. □

The above proposition implies a sufficient condition which guarantees the consistency of $V(P)$ for a locally stratified program P.

Corollary 3.9 *Let P be a locally stratified program. If P satisfies the condition (i) (resp. (ii)) of Proposition 3.8, $A(P)$ (resp. $I(P)$) is consistent under both the stable model semantics and the completion semantics.*

Proof. When P satisfies the condition (i) (resp. (ii)), $A(P)$ (resp. $I(P)$) is locally stratified. Since any locally stratified program is consistent under both the stable model semantics and the completion semantics, the result holds. □

4 Procedural Properties

In the previous section, we observed that the V-operators may introduce cycles to a program. This means that given an acyclic program P, the completeness of SLDNF-resolution in the produced program $V(P)$ is not guaranteed in general. The conditions of Proposition 3.8 are useful for keeping $V(P)$ acyclic. This problem does not happen when a program is definite. When a definite program contains variables, however, an application of the V-operators may lose answers which are computed using SLD-resolution in the original program.

Example 4.1. Let P_1 be the program

$$p(x, y) \leftarrow q(x, y), \quad r(y) \leftarrow q(x, y), \quad q(a, b) \leftarrow,$$

and $A(P_1)$ the program

$$p(x, y) \leftarrow r(y), \quad r(y) \leftarrow q(x, y), \quad q(a, b) \leftarrow .$$

Using SLD-resolution, the query $\leftarrow p(x, b)$ computes the answer $x = a$ in P_1, but the answer is not obtained in $A(P_1)$. Next, let P_2 be the program

$$p(x, y) \leftarrow r(y), \quad p(x, y) \leftarrow q(x, y), \quad q(a, b) \leftarrow,$$

and $I(P_2)$ the program

$$p(x,y) \leftarrow r(y), \quad r(y) \leftarrow q(x,y), \quad q(a,b) \leftarrow .$$

Using SLD-resolution, the query $\leftarrow p(x,b)$ computes the answer $x = a$ in P_2, but the answer is not obtained in $I(P_2)$.

Thus, unrestricted application of the V-operators does not always extend the set of computed answers even in a definite program. Note that such phenomena happen only when the input rule C_3 is included in the original program P in (2) and (3). When C_3 is given aside from P, the relation $P \subseteq V(P)$ holds hence every computed answer in P is obtained in $V(P)$. Hence, the following arguments are meaningful when C_3 is in P.

We first define the notion of generalization wrt computed answers. Given a definite program P and a goal $\leftarrow G$, we write $P \models_{SLD} G\theta$ if the goal has an SLD-refutation with a computed answer θ.[5]

Definition 4.1. Let P_1 and P_2 be definite programs. Then, P_1 is a *generalization of P_2 wrt computed answers* if $P_2 \models_{SLD} G$ implies $P_1 \models_{SLD} G$ for any atom G.

The next proposition presents sufficient conditions for the V-operators to generalize a program wrt computed answers.

Proposition 4.1. *Let P be a definite program.*

(i) $A(P)$ is a generalization of P wrt computed answers if in the rule $C_1 : q \leftarrow A$, every variable in A appears in q.
(ii) $I(P)$ is a generalization of P wrt computed answers if in the rule $C_2 : p \leftarrow q, B$, every variable in p appears in either q or B.

Proof. (i) Without loss of generality, we can put $C_1 : q(x,y) \leftarrow A(x)$ and $C_3 : p(z) \leftarrow A(x), B(w)$, where x, y, z, w are vectors of terms. Suppose that $C_2 : p(z) \leftarrow q(x,y), B(w)$ is produced by absorption. Resolving C_2 with C_1, we get the original rule C_3 in $A(P)$. Hence, if a query has a computed answer in P, the same answer is computed by SLD-resolution in $A(P)$.

(ii) Put $C_2 : p(x,y) \leftarrow q(x,z), B(y,w)$ and $C_3 : p(x,y) \leftarrow A(u), B(y,w)$, where x, y, z, w, u are vectors of terms. Suppose that $C_1 : q(x,z) \leftarrow A(u)$ is produced by identification. Resolving C_2 with C_1, we get the rule $C_3' : p(x,y) \leftarrow A(u'), B(y,w)$ where u' is possibly different from u. For variables in x, two cases are considered. (a) When x and u share no variable in C_3, x and u' also share no variable in C_3'. (b) When x and u share variables in C_3, the same variables are shared in x and u in C_1 and thereby shared in x and u' in C_3'. In either case, any binding for the variables in x, that is computed using C_3 in P, is also computed using C_3' in $I(P)$. Next, for variables in y, two cases are considered. (c) When y and u share no variable in C_3, y and u' also share no variable in C_3'.

[5] We assume familiarity with basic terminologies wrt SLD-resolution given in [9].

(d) When y and u share variables in C_3, the variables are possibly renamed in u' in C_3' but any binding for the variables in y is computed by $B(y, w)$ in C_3'. In either case, any binding for the variables in y, that is computed using C_3 in P, is also computed using C_3' in $I(P)$. Therefore, if a query has a computed answer in P, the same answer is computed by SLD-resolution in $I(P)$. □

The V-operators do not preserve the condition of *allowedness*[6] in general. For instance, the program P_1 of Example 4.1 is allowed but $A(P_1)$ is not. In normal logic programs, the condition of allowedness provides a sufficient condition to prevent a query from *floundering* [9]. Thus, given an allowed normal logic program P, a query may flounder in $V(P)$. The condition (i) of Proposition 4.1 prevents the decrease of variables in the body of C_3, hence it guarantees that $A(P)$ is allowed if P is allowed. On the other hand, the condition (ii) is insufficient to keep $I(P)$ allowed. Suppose that a program P has the rules

$$C_2 : p(x, y) \leftarrow q(x, y),\ s(y), \quad C_3 : p(x, y) \leftarrow r(x),\ s(y),$$

which are allowed and C_2 satisfies the condition (ii) of Proposition 4.1. But identification produces $C_1 : q(x, y) \leftarrow r(x)$ which is not allowed. Then, the query $\leftarrow q(x, y), \neg t(x, y)$ may flounder in $I(P)$. To keep $I(P)$ allowed, it is sufficient that every variable in q of C_2 appears in a positive literal in A of C_3 in P.

5 Discussion

There are variants of the V-operators. Muggleton [12] introduces the *most specific* version of the V-operators. The most specific absorption produces the rule $C_2' : p \leftarrow q, A, B$, instead of $C_2 : p \leftarrow q, B$, while the most specific identification produces the rule $C_1' : q \leftarrow A, B$, instead of $C_1 : q \leftarrow A$. The most specific absorption is also called *saturation* in [16]. Saturation alone does not generalize a program and is followed by *truncation*. Truncation drops the conjunction A from C_2', which results in C_2. Coupling saturation and truncation includes absorption as a special case, hence they have the same properties as those of absorption presented in this paper. It is easy to see that the properties of identification presented in this paper also hold for the most specific identification. The *W-operators* of [13] include the V-operators as a special case, hence they also inherit the properties of the V-operators.

There are few work which considers inverse resolution in normal logic programs. Bain and Muggleton [3, 4] incorporate the *closed world specialization* technique into CIGOL, but they do not provide formal analysis of inverse resolution in nonmonotonic theories. Taylor [17] introduces *normal absorption* which is different from absorption in definite programs. Given the input rules $p \leftarrow q$ and $r \leftarrow q, \neg s$, normal absorption outputs the rule $p \leftarrow r, [q, \neg s]$ where $[q, \neg s]$ presents optional literals which can be dropped at the end. She shows that the output rule generalizes the input rule wrt the background theory under normal subsumption. However, she does not argue the effect of such normal V-operators in a whole theory.

[6] Any variable in a rule has an occurrence in a positive literal in the body of the rule.

6 Summary

This paper studied the effect of inverse resolution in normal logic programs. We posed several problems of the V-operators that may occur in the presence of negation as failure, and gave some sufficient conditions to avoid these problems. The results of this paper notice that care should be taken when using the V-operators as inductive operations in normal logic programs.

Acknowledgements

The author thanks Katsumi Inoue for comments on an earlier draft of this paper.

References

1. K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In: *Foundations of Deductive Databases and Logic Programming* (J. Minker ed.), Morgan Kaufmann, pp. 89–148, 1988.
2. K. R. Apt and M. Bezem. Acyclic programs. *New Generation Computing* 9:335–363, 1991.
3. M. Bain and S. Muggleton. Non-monotonic learning. In: [14], pp. 145–161.
4. M. Bain. Experiments in non-monotonic first-order induction. In: [14], pp. 423–436.
5. R. B. Banerji. Learning theoretical terms. In: [14], pp. 93–112.
6. C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming* 19/20:73–148, 1994.
7. K. L. Clark. Negation as failure. In: H. Gallaire and J. Minker (eds.), *Logic and Data Bases*, Plenum Press, pp. 119–140, 1978.
8. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In: *Proc. 5th Int'l Conf. and Symp. on Logic Programming*, MIT Press, pp. 1070–1080, 1988.
9. J. W. Lloyd. *Foundations of logic programming* (2nd edition), Springer-Verlag, 1987.
10. S. Muggleton. Duce, an oracle based approach to constructive induction. In: *Proc. IJCAI-87*, Morgan Kaufmann, pp. 287–292, 1987.
11. S. Muggleton. Inverting the resolution principle. In: *Machine Intelligence*, vol. 12, Oxford University Press, pp. 93–103, 1991.
12. S. Muggleton. Inductive Logic Programming. In [14], pp. 3–27, 1992.
13. S. Muggleton and W. Buntine. Machine invention of first-order predicate by inverting resolution. In: [14], pp. 261–280.
14. S. Muggleton (ed.). *Inductive Logic Programming*, Academic Press, 1992.
15. T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann, pp. 193–216, 1988.
16. C. Rouveirol. Extension of inversion of resolution applied to theory completion. In: [14], pp. 63–92.
17. K. Taylor. Inverse resolution of normal clauses. In: *Proc. ILP-93*, J. Stefan Institute, pp. 165–177, 1993.