Inverse Entailment in Nonmonotonic 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. Inverse entailment (IE) is known as a technique for finding inductive hypotheses in Horn theories. When a background theory is nonmonotonic, however, IE is not applicable in its present form. The purpose of this paper is extending the IE technique to nonmonotonic inductive logic programming (ILP). To this end, we first establish a new entailment theorem in normal logic programs, then introduce the notion of contrapositive programs. Finally, a theory of IE in nonmonotonic ILP is constructed.

1 Introduction

Inverse entailment (IE) [11] is one of the basic techniques in inductive logic programming (ILP), which is used for computing inductive hypotheses in the following manner. Given a background Horn logic program B and an example E as a Horn clause, suppose a hypothetical Horn clause H satisfying

$$B \wedge H \models E.$$

By inverting the entailment relation it becomes

$$B \wedge \neg E \models \neg H.$$

Put $\neg\mathsf{Bot}$ as the conjunction of ground literals which are true in every model of $B \wedge \neg E$. Consider the case of $\neg\mathsf{Bot} \models \neg H$.[1] Then, it becomes $H \models \mathsf{Bot}$. Such Bot is effective for reducing the hypothesis space, since a possible hypothesis H is constructed as a clause which subsumes Bot.

A Horn logic program is *monotonic* in the sense that adding a clause to the program never leads to the loss of any conclusions previously proved in the program. However, it is known that Horn logic programs are not sufficiently expressive for the representation of *incomplete knowledge*. In the real world, humans perform *commonsense reasoning* when one's knowledge is incomplete.

[1] Note that $B \wedge \neg E \models \neg H$ does not imply $\neg\mathsf{Bot} \models \neg H$. This is the reason for the incompleteness of IE [15].

Commonsense reasoning is *nonmonotonic* in its feature, that is, previously concluded facts might be withdrawn by the introduction of new information. A logic program which has the nonmonotonic feature is called a *nonmonotonic logic program*.

When a background theory is a nonmonotonic logic program, however, the above IE relation does not hold in general. The IE relation is derived from the original entailment as follows:

$$B \wedge H \models E \;\Leftrightarrow\; B \models (H \rightarrow E) \quad (1)$$
$$\Leftrightarrow\; B \models (\neg E \rightarrow \neg H) \quad (2)$$
$$\Leftrightarrow\; B \wedge \neg E \models \neg H. \quad (3)$$

When B is nonmonotonic, there are two problems in the above derivation process. First, in nonmonotonic logic the deduction theorem does not hold in general [14], hence the equivalence relations (1) and (3) do not hold. Second, in nonmonotonic logic programs, a rule presents a derivation rule in one-way direction, so that the contrapositive implication (2) is undefined. These facts mean that the present IE technique cannot be used when a background theory is nonmonotonic. Then, reconstruction of the framework is necessary to apply IE to nonmonotonic ILP.

This paper studies a theory of inverse entailment in *nonmonotonic ILP*. As nonmonotonic logic programs, we consider *normal logic programs*, i.e., logic programs with *negation as failure*. Negation as failure represents default negation in a program, which makes the semantics of programs different from classical logic. We first review the problem of the deduction theorem in nonmonotonic theories, then introduce the new entailment theorem in normal logic programs. Next, we introduce contrapositive rules in normal logic programs and present their semantic properties. Finally, we construct a theory of inverse entailment in normal logic programs.

The rest of this paper is organized as follows. Section 2 presents a theoretical framework used in this paper. Section 3 establishes the new entailment theorem in normal logic programs. Section 4 introduces a framework of contrapositive programs. Section 5 constructs a theory of inverse entailment in normal logic programs, and Section 6 provides examples. Section 7 discusses related issues and Section 8 concludes the paper.

2 Normal Logic Programs

Definitions of basic notions such as *constants*, *functions*, *variables*, *predicates*, and *atoms* follow from those standard in logic programming [9]. A logic program considered in this paper is a *normal logic program* [9, 2], which is a set of rules of the form:

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n \quad (4)$$

where each A_i $(0 \leq i \leq n)$ is an atom and *not* presents *negation as failure* (NAF). The atom A_0 is the *head*, and the conjunction $A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n$

is the *body* which is identified with the set $\{ A_1, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n \}$. We allow a rule with an empty head of the form:

$$\leftarrow A_1, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n \tag{5}$$

which is also called an *integrity constraint*. A rule with an empty body $A \leftarrow$ is identified with the atom A. Throughout the paper a program means a normal logic program unless stated otherwise. A program P is *Horn* if no rule in P contains NAF. The *Herbrand base* $\mathcal{HB}$ of a program P is the set of all ground atoms in the language of P. Given the Herbrand base $\mathcal{HB}$, we define $\mathcal{HB}^+ = \mathcal{HB} \cup \{ not\, A \mid A \in \mathcal{HB} \}$. Any element in $\mathcal{HB}^+$ is called an *LP-literal*. A program, a rule, or an LP-literal is *ground* if it contains no variable. Any variable in a program is interpreted as a free variable. In this case, 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 with elements of the Herbrand universe in every possible way.

An *interpretation* is a subset of the Herbrand base of a program. An interpretation I *satisfies* the conjunction of ground LP-literals $C = A_1, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n$ if $\{A_1, \ldots, A_m\} \subseteq I$ and $\{A_{m+1}, \ldots, A_n\} \cap I = \emptyset$ (written as $I \models C$). An interpretation I satisfies the ground rule R of the form (4) if $I \models A_1, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n$ implies $A_0 \in I$ (written as $I \models R$). In particular, I satisfies the ground integrity constraint of the form (5) if either $\{A_1, \ldots, A_m\} \setminus I \neq \emptyset$ or $\{A_{m+1}, \ldots, A_n\} \cap I \neq \emptyset$. When a rule R contains variables, $I \models R$ means that I satisfies every ground instance of R. When I does not satisfy R (i.e., $I \not\models R$), it is also written as $I \models not\, R$. We define $I^+ \models R$ if $I \models R$ where $I^+ = I \cup \{ not\, A \mid A \in \mathcal{HB} \setminus I \}$. An interpretation which satisfies every rule in a program is a *model* of the program. A model M of a program P is *minimal* if there is no model N of P such that $N \subset M$.

A Horn logic program has at most one minimal model, i.e., the *least model*, which provides the declarative meaning of a program. On the other hand, a normal logic program generally has multiple minimal models. For instance, the program

$$a \leftarrow not\, b$$

has two minimal models $\{a\}$ and $\{b\}$. However, in this program the rule asserts that a is true if b is not true. Since there is no evidence of b, it is natural to conclude a in the program. In this regard, the minimal model $\{a\}$ should be taken as the intended meaning of the program. Thus, in normal logic programs it is necessary to identify the class of minimal models which provide the intended meaning of a program. The *stable model semantics* proposed by Gelfond and Lifschitz [6] is one of the most popular semantics of normal logic programs. A stable model provides a natural meaning for many normal logic programs and coincides with the least model in Horn logic programs. Moreover, the stable model semantics plays an important role to relate logic programming and nonmonotonic formalisms in AI [2].

A formal definition of the stable model semantics is as follows. Let R be a ground rule of the form (4) or (5) and I an interpretation. Then, define R^I as

$$R^I = \begin{cases} A_0 \leftarrow A_1, \ldots, A_m & \text{if } \{A_{m+1}, \ldots, A_n\} \cap I = \emptyset. \\ \mathit{true}, & \text{otherwise.} \end{cases}$$

Given a program P and an interpretation I, the ground Horn program P^I is defined as

$$P^I = \{\, R^I \mid R \text{ is a rule in the ground instantiation of } P \,\}.$$

If the least model of P^I is identical to I, I is called a *stable model* of P.

The intuitive meaning of the transformation from P to P^I is as follows. First assume an interpretation I as a possible set of beliefs. Then, for any ground rule $R:\ A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n$, if $\{A_{m+1}, \ldots, A_n\} \cap I = \emptyset$, then the conjunction $\mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n$ is true wrt I and R is simplified as R^I in P^I. Otherwise, R contains some $\mathit{not}\, A_i$ $(m+1 \leq i \leq n)$ such that $A_i \in I$, then I does not satisfy the condition of R and such R is useless for derivation thereby deleted (or replaced by true) in P^I. If the least model of P^I coincides with I, the belief set I is justified and is called a stable model.

A program may have none, one, or multiple stable models in general. A program having exactly one stable model is called *categorical* [2].[2] A stable model is a minimal model and it coincides with the least model in a Horn logic program. A program is *consistent* (under the stable model semantics) if it has a stable model; otherwise a program is *inconsistent*. Any program is assumed to be consistent in this paper.

Example 2.1. The program

$$a \leftarrow \mathit{not}\, b,$$
$$b \leftarrow \mathit{not}\, c$$

has the unique stable model $\{b\}$. Hence, the program is categorical.

The program

$$a \leftarrow \mathit{not}\, b,$$
$$b \leftarrow \mathit{not}\, a$$

has two stable models $\{a\}$ and $\{b\}$, which represents two alternative beliefs.

On the other hand, the program

$$a \leftarrow \mathit{not}\, a$$

has no stable model, hence the program is inconsistent.

If every stable model of a program P satisfies a rule R, it is written as $P \models_s R$. Else if no stable model of a program P satisfies a rule R, it is written as $P \models_s \mathit{not}\, R$. In particular, $P \models_s A$ if a ground atom A is true in every stable model of P; and $P \models_s \mathit{not}\, A$ if A is false in every stable model of P. In contrast, if every model of P satisfies R, it is written as $P \models R$. Note that when P is Horn, the meaning of $\models$ coincides with the classical entailment.

[2] An example of the class of programs with this property is a *locally stratified program*.

3 Entailment Theorem

In classical first-order logic, the following deduction theorem holds.

> Given a set of formulas T and a closed formula F,
> $T \wedge F \models G$ iff $T \models F \rightarrow G$ for any formula G.

In the context of *nonmonotonic logic* (NML), Shoham [14] shows that the above theorem does not hold in general. He argues that this is due to the difference of the definitions of entailment between classical logic and nonmonotonic logic. In classical logic, a formula F is entailed by a theory T (i.e., $T \models F$) if F is satisfied in every model of T. In nonmonotonic logic, on the other hand, the corresponding relation $T \models_{NML} F$ is defined if F is satisfied in every *preferred* model of T. The set of preferred models of T is a subset of the set of all models of T, then a formula F which cannot be entailed in classical logic might be entailed in nonmonotonic logic.[3]

In this paper, we regard stable models as preferred models of a program. Then, the next relation holds.

Proposition 3.1 *Let P be a normal logic program. For any rule R, $P \models R$ implies $P \models_s R$, but not vice-versa.*

Proof. $P \models R$ means that R is satisfied in every model of P. Since stable models are models of P, $P \models R$ implies $P \models_s R$. It is enough to show a counter-example for the converse. Let $P = \{\, a \leftarrow not\, b \,\}$ which has the unique stable model $\{a\}$. Then, $P \models_s a$ but $P \not\models a$. This is because P has the (non-stable) model $\{b\}$ in which a is false. □

Note that the converse does not hold in general even if a program is Horn.

Example 3.1. Let P be the program

$$a \leftarrow b$$

which has the stable model (or the least model) $\emptyset$. Then, $P \models_s c \leftarrow d$ but $P \not\models c \leftarrow d$. That is, $c \leftarrow d$ is satisfied in the least model of P, but it is not necessarily satisfied in every model of P. In fact, P has the model $\{d\}$ which does not satisfy $c \leftarrow d$.

Shoham presents that in nonmonotonic logic the deduction theorem is false in general, while the only-if direction still holds.[4] We first provide the corresponding result in normal logic programs. A *nested rule* is defined as

$$A \leftarrow R$$

where A is an atom and R is a rule of the form (4). An interpretation I satisfies a ground nested rule $A \leftarrow R$ if $I \models R$ implies $A \in I$. For a program P, $P \models_s (A \leftarrow R)$ if $A \leftarrow R$ is satisfied in every stable model of P.

[3] This characterizes the unique feature of NML that "jumping to a conclusion".
[4] Shoham provides no proof of this fact.

Theorem 3.2. *Let P be a normal logic program and R a rule such that $P \cup \{R\}$ is consistent. If $P \cup \{R\} \models_s A$, then $P \models_s A \leftarrow R$ for any ground atom A.*

Proof. When $P \cup \{R\} \models_s A$, A is true in every stable model of $P \cup \{R\}$. To see $P \models_s A \leftarrow R$, we show that for any stable model M of P, $M \models R$ implies $A \in M$. Suppose that there is a stable model M of P such that $M \models R$. Let R_g be any ground instance of R. Clearly, $M \models R$ implies $M \models R_g$, thereby $M \models {R_g}^M$. Since M is the least model of P^M and $M \models {R_g}^M$, M is a model of $(P^M \cup \{{R_g}^M\}) = (P \cup \{R\})^M$. Suppose that N is the least model of $(P \cup \{R\})^M$ s.t. $N \subseteq M$. As $(P \cup \{R\})^M$ is Horn and consistent, the least model of P^M becomes a subset of the least model of $(P \cup \{R\})^M$, i.e., $M \subseteq N$. Hence, $M = N$. Since M is the least model of $(P \cup \{R\})^M$, M is a stable model of $P \cup \{R\}$. As A is true in every stable model of $P \cup \{R\}$, $A \in M$ holds. Hence, $M \models R$ implies $A \in M$, and the result holds. □

The converse does not hold in general.

Example 3.2. Let P be the program

$$a \leftarrow not\, b$$

which has the unique stable model $\{a\}$. Then, $P \models_s a \leftarrow b$. On the other hand, $P \cup \{b\}$ has the unique stable model $\{b\}$, so $P \cup \{b\} \not\models_s a$.

An additional condition is necessary for the converse.

Theorem 3.3. *Let P be a normal logic program and R a rule such that $P \cup \{R\}$ is consistent. Then, if $P \models_s A \leftarrow R$ and $P \models_s R$, then $P \cup \{R\} \models_s A$ for any ground atom A.*

Proof. By $P \models_s R$, $M \models R$ for any stable model M of P. In this case, $P \models_s A \leftarrow R$ implies $A \in M$. Then, for any ground instance R_g of R, $M \models {R_g}^M$ holds. Since M is the least model of P^M and $M \models {R_g}^M$, M is a model of $(P^M \cup \{{R_g}^M\}) = (P \cup \{R\})^M$. Suppose that N is the least model of $(P \cup \{R\})^M$ s.t. $N \subseteq M$. Then, we can show $M = N$ in the same manner as Theorem 3.2. Thus, $A \in M$ for the least model M of $(P \cup \{R\})^M$. This implies that $A \in M$ for any stable model M of $P \cup \{R\}$. Hence, $P \cup \{R\} \models_s A$. □

We call Theorems 3.2 and 3.3 the *entailment theorem* of normal logic programs.[5]

4 Contrapositive Rules

In normal logic programs, a rule in a program is not a "clause" in the classical sense because of the presence of negation as failure. This makes the meaning of

[5] We avoid using the term "deduction" because the entailment symbol $\models_s$ does not imply deductive consequences in the classical sense.

rules different from the one in classical logic. One of such distinctive features is that a rule and its *contrapositive* with respect to the implication $\leftarrow$ and negation *not* are not semantically equivalent. For example, the program

$$a \leftarrow$$

has the stable model $\{a\}$, while the program which consists of its contrapositive

$$\leftarrow \mathit{not}\, a$$

has no stable model.

However, in the process of inverse entailment a rule must be contraposed, hence the introduction of contrapositive rules is necessary to apply inverse entailment to normal logic programs. This section introduces a theory of contrapositive rules in normal logic programs and presents their properties.

Definition 4.1. Given a rule R of the form:

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n\,,$$

its *contrapositive rule* ${}^cR$ is defined as

$$\mathit{not}\, A_1\,; \cdots ; \mathit{not}\, A_m; \mathit{not}\, \mathit{not}\, A_{m+1}\,; \cdots ; \mathit{not}\, \mathit{not}\, A_n \leftarrow \mathit{not}\, A_0 \qquad (6)$$

where ";" means disjunction. The left-hand side of $\leftarrow$ is the *head* and the right-hand side is the *body* of the rule.

In the contrapositive rule, the atoms $A_0, A_1, \ldots, A_m$ in the original rule are shifted to the other side of the implication with the introduction of *not*. On the other hand, the NAF formulas $\mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n$ in the body are shifted to the head of the rule with the introduction of "nested NAF". The semantics of such nested NAF is given in [8]. A ground rule of the form (6) is *satisfied* in an interpretation I if $A_0 \notin I$ implies either $\{A_1, \ldots, A_m\} \setminus I \neq \emptyset$ or $\{A_{m+1}, \ldots, A_n\} \cap I \neq \emptyset$ (written as $I \models {}^cR$). When a rule ${}^cR$ contains variables, $I \models {}^cR$ means that I satisfies every ground instance of ${}^cR$.

According to [8], the rule (6) is equivalent to the integrity constraint

$$\leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n, \mathit{not}\, A_0 \qquad (7)$$

under the stable model semantics. Hence we identify the contrapositive rule (6) with the integrity constraint (7) hereafter. In particular,

$$\mathit{not}\, A \leftarrow$$

is identified with

$$\leftarrow A,$$

and

$$\mathit{not}\, \mathit{not}\, A \leftarrow$$

is identified with

$$\leftarrow \mathit{not}\, A.$$

The contrapositive form of a nested rule is defined as follows. A *nested rule* R considered here is of the form:

$$L \leftarrow Q$$

where L is an LP-literal and Q is a rule of the form (4). An interpretation I satisfies a ground nested rule $L \leftarrow Q$ if $I \models Q$ implies $I \models L$. For a program P, $P \models_s (L \leftarrow Q)$ if $L \leftarrow Q$ is satisfied in every stable model of P. Then, a *contrapositive rule* ${}^cR$ is defined as

$$\mathit{not}\, Q \leftarrow \mathit{not}\, L\,. \tag{8}$$

A ground rule of the form (8) is *satisfied* in an interpretation I if $I \not\models L$ implies $I \not\models Q$. In particular, a ground rule of the form

$$\mathit{not}\, Q \leftarrow$$

is satisfied in I if $I \not\models Q$.

For a program P, $P \models_s {}^cR$ if ${}^cR$ is satisfied in every stable model of P.

Theorem 4.1. *Let P be a normal logic program, R a (nested) rule, and ${}^cR$ its contrapositive. Then, $P \models_s R$ iff $P \models_s {}^cR$.*

Proof. $P \models_s R$ iff $P \models_s R_g$ for any ground instance R_g of R.
First, let R_g be a ground (unnested) rule of the form
$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n$. Then, $P \models_s R_g$
iff for any stable model M of P, $\{A_1, \ldots, A_m\} \subseteq M$ and $\{A_{m+1}, \ldots, A_n\} \cap M = \emptyset$ imply $A_0 \in M$
iff for any stable model M of P, $A_0 \notin M$ implies either $\{A_1, \ldots, A_m\} \setminus M \neq \emptyset$ or $\{A_{m+1}, \ldots, A_n\} \cap M \neq \emptyset$
iff $P \models_s {}^cR_g$
iff $P \models_s {}^cR$.

Next, let R_g be a ground nested rule of the form $L \leftarrow Q$. Then, $P \models_s R_g$
iff for any stable model M of P, $M \models Q$ implies $M \models L$
iff for any stable model M of P, $M \not\models L$ implies $M \not\models Q$
iff $P \models_s {}^cR_g$
iff $P \models_s {}^cR$. □

5 Inverse Entailment in Normal Logic Programs

We first apply the entailment theorem (Theorems 3.2 and 3.3) to a ground LP-literal $\mathit{not}\, A$.

Theorem 5.1. *Let P be a normal logic program and R a rule such that $P \cup \{R\}$ is consistent. If $P \cup \{R\} \models_s \mathit{not}\, A$, then $P \models_s (\mathit{not}\, A \leftarrow R)$ for any ground atom A. In converse, if $P \models_s (\mathit{not}\, A \leftarrow R)$ and $P \models_s R$, then $P \cup \{R\} \models_s \mathit{not}\, A$.*

Proof. If $P \cup \{R\} \models_s not\ A$, A is false in every stable model of $P \cup \{R\}$. Then, the rest of the proof in this direction proceeds in the same manner as Theorem 3.2. On the other hand, if $P \models_s (not\ A \leftarrow R)$ and $P \models_s R$, $A \notin M$ for any stable model M of P. Then, for any ground instance R_g of R, $M \models {R_g}^M$ and $A \notin M$ for the least model M of P^M. Then, we can show that M is the least model of $(P \cup \{R\})^M$ as in Theorem 3.3. This implies that $A \notin M$ for any stable model M of $P \cup \{R\}$. Hence, $P \cup \{R\} \models_s not\ A$. □

Proposition 5.2 *Let P be a normal logic program and L a ground LP-literal. If $P \models_s not\ L$, then $P \cup \{\ not\ L\ \}$ is consistent.*[6]

Proof. $P \models_s not\ L$ implies $P \models_s \leftarrow L$. Then, $M \models \leftarrow L$ holds for every stable model M of P. Let M be any stable model of P. Then, M is the least model of P^M and $M \models \leftarrow L$ iff M is the least model of $(P \cup \{\leftarrow L\})^M$ iff M is a stable model of $P \cup \{\leftarrow L\}$. Hence, $P \cup \{\ not\ L\ \}$ is consistent. □

Now we are ready to introduce inverse entailment in normal logic programs.

Theorem 5.3. *Let P be a normal logic program and R a rule such that $P \cup \{R\}$ is consistent. For any ground LP-literal L, if $P \cup \{R\} \models_s L$ and $P \models_s not\ L$, then $P \cup \{\ not\ L\ \} \models_s not\ R$ where $P \cup \{\ not\ L\ \}$ is consistent.*

Proof. If $P \cup \{R\} \models_s L$, then $P \models_s (L \leftarrow R)$ (Theorems 3.2 and 5.1). On the other hand, $P \models_s (L \leftarrow R)$ iff $P \models_s (not\ R \leftarrow not\ L)$ (Theorem 4.1). Put $A = R$, where A is an atom $p(x)$ such that p appears nowhere in P and x is a vector of variables appearing in R. Then, $P \models_s (not\ R \leftarrow not\ L)$ iff $P \models_s (not\ A \leftarrow not\ L)$. By $P \models_s (not\ A \leftarrow not\ L)$ and $P \models_s not\ L$, it holds that $P \cup \{\ not\ L\ \} \models_s not\ A$ (Theorems 5.1). Thus, $P \cup \{\ not\ L\ \} \models_s not\ R$. Also, $P \models_s not\ L$ implies the consistency of $P \cup \{\ not\ L\ \}$ (Proposition 5.2). Hence, the result holds. □

The converse of Theorem 5.3 does not hold in general.

Example 5.1. Let P be the program

$$a \leftarrow b$$

with $L = c$, and $R = (b \leftarrow)$. In this case, $P \cup \{\ not\ L\ \}$ becomes

$$a \leftarrow b,$$
$$\leftarrow c,$$

where $not\ c$ is expressed as the constraint $\leftarrow c$. This program has the stable model $\emptyset$ in which R is false, so that $P \cup \{\ not\ L\ \} \models_s not\ R$. On the other hand, $P \cup \{R\}$ becomes

$$a \leftarrow b,$$
$$b \leftarrow,$$

which has the stable model $\{\ a, b\ \}$. Since $L = c$ is false in this stable model, $P \cup \{R\} \not\models_s L$.

[6] $\{\ not\ L\ \}$ is an abbreviation of $\{\ not\ L \leftarrow\ \}$ which is equivalent to $\{\leftarrow L\}$.

Thus, the relation $P \cup \{ not\, L \} \models_s not\, R$ provides a necessary (but not always sufficient) condition for computing a rule R satisfying $P \cup \{R\} \models_s L$ and $P \models_s not\, L$.

Next we present a method of applying the above new inverse entailment theorem to computing inductive hypotheses in normal logic programs. We first characterize an induction problem considered here.

Given :
- a *background knowledge base* P as a *categorical* normal logic program,
- a ground LP-literal L such that $P \models_s not\, L$, where L represents a *positive example* in case of $L = A$ with a ground atom A; otherwise it represents a *negative example* in case of $L = not\, A$,
- a *target predicate* which is subject to learn.

Find : a hypothetical rule R such that
- $P \cup \{R\} \models_s L$,
- $P \cup \{R\}$ is consistent,
- R has the target predicate in its head.

Here, we assume that P, R, and L have the same language. In the above, the assumption $P \models_s not\, L$ presents that the given example is false in the unique stable model of the background program.[7] In fact, if the example L is true in P, there is no need to find R.

First, suppose that there is a rule R such that $P \cup \{R\}$ is consistent and $P \cup \{R\} \models_s L$. When $P \models_s not\, L$, it holds by Theorem 5.3 that

$$P \cup \{ not\, L \} \models_s not\, R \tag{9}$$

where $P \cup \{ not\, L \}$ is consistent.

The relation (9) means that R is not satisfied in the stable model of $P \cup \{ not\, L \}$. Here we assume that P has the unique stable model and $P \models_s not\, L$. Then, the next proposition holds.

Proposition 5.4 *Let P be a categorical normal logic program. Also, let L be a ground LP-literal such that $P \models_s not\, L$. Then, the stable model of P is equivalent to the stable model of $P \cup \{ not\, L \}$.*

Proof. M is the stable model of P s.t. $P \models_s not\, L$
iff M is the least model of P^M and $M \models \leftarrow L$
iff M is the least model of $(P \cup \{ \leftarrow L \})^M$
iff M is the stable model of $P \cup \{ \leftarrow L \}$
iff M is the stable model of $P \cup \{ not\, L \}$. □

Using Proposition 5.4, the relation (9) is simplified as

$$P \models_s not\, R\,. \tag{10}$$

[7] If $L = not\, A$, $P \models_s not\, not\, A$ means that A is true in the stable model of P.

Next, put

$$M^+ = \{\, l \mid l \in \mathcal{HB}^+ \text{ and } P \models_s l \,\}.$$

M^+ is the set of LP-literals, which augments the stable M of P by the LP-literals $\{\, not\, A \mid A \in \mathcal{HB} \setminus M \,\}$. We call M^+ the *expansion* of M. Then, by (10) it holds that

$$M^+ \models not\, R\,. \tag{11}$$

Let r_0 be an integrity constraint $\leftarrow \Gamma$ where Γ is a conjunction of ground LP-literals such that $\Gamma \subseteq M^+$. Since M^+ does not satisfy r_0, r_0 satisfies the relation (11), i.e., $M^+ \models not\, r_0$.

Here we define a couple of notions. Given two ground LP-literals L_1 and L_2, we define the relation $L_1 \sim L_2$ if L_1 and L_2 have the same predicate and $const(L_1) = const(L_2)$ where $const(L)$ denotes the set of constants appearing in L. Let L_1 and L_2 be two ground LP-literals such that each literal has a predicate with more than 0 argument. Then, L_1 in a ground rule R is *relevant* to L_2 if either (i) $L_1 \sim L_2$ or (ii) L_1 shares a constant with an LP-literal L_3 in R such that L_3 is relevant to L_2. Otherwise, L_1 is *irrelevant* to L_2.

Now we apply the following series of transformations to the above r_0.

1. (dropping irrelevant atoms): Drop any LP-literal from r_0 which is irrelevant to the given example. The resulting rule is denoted as r_1.
2. (dropping implied atoms): Suppose that the body of r_1 contains both an atom A and the conjunction B of LP-literals such that $A \neq B$, and the rule $A \leftarrow B$ is in the ground instantiation of the background program. Then, drop A from the body of r_1. The resulting rule is denoted as r_2.
3. (shifting the target atom): Let $r_2 = \leftarrow \Gamma$ where Γ is a conjunction of ground LP-literals. If Γ contains an LP-literal $not\, A$ with the target predicate, produce the rule $r_3 : A \leftarrow \Gamma'$ where $\Gamma' = \Gamma \setminus \{not\, A\}$.
4. (generalization): Replace constants appearing in r_3 by appropriate variables. That is, construct a rule r_4 such that $r_4\theta = r_3$ for some substitution θ.

The above transformation sequence 1–4 is denoted as $Trans$. The next theorem presents that any rule produced by $Trans$ satisfies the relation (11).

Theorem 5.5. *Let M^+ be a set of ground LP-literals defined as above. If R_{ie} is a rule which is obtained by $Trans$, then $M^+ \models not\, R_{ie}$ holds.*

Proof. Let $M^+ \models not\, r_0$. Dropping any ground LP-literals from the body of the constraint r_0 does not violate this relation. Hence, $M^+ \models not\, r_1$ and $M^+ \models not\, r_2$. Next, let $r_3 : A \leftarrow \Gamma'$ which is obtained from r_2 by converting $not\, A$ in the body of r_2 to A in the head of r_3. By $M^+ \models not\, r_2$, $\Gamma' \subseteq M^+$ and $not\, A \in M^+$. Hence, $M^+ \models not\, r_3$. Finally, consider r_4. Since M^+ does not satisfy a ground instance r_3 of r_4, $M^+ \not\models r_4$ thereby $M^+ \models not\, r_4$. Hence, the result holds. □

When $P \cup \{R_{ie}\}$ is consistent, we say that a rule R_{ie} is computed using the *inverse entailment rule* (*IE rule*, for short) from an LP-literal L in the program P (under the stable model semantics).

6 Examples

Example 6.1. Let P be the background program:

$$\begin{aligned} &bird(x) \leftarrow penguin(x),\\ &bird(tweety) \leftarrow,\\ &penguin(polly) \leftarrow . \end{aligned}$$

Given the positive example $L = flies(tweety)$, $P \models_s not\, flies(tweety)$. Then, the set M^+ of LP-literals becomes

$$\begin{aligned} M^+ = \{ &bird(tweety),\, bird(polly),\, penguin(polly),\\ &not\, penguin(tweety),\, not\, flies(tweety),\, not\, flies(polly)\,\}. \end{aligned}$$

From M^+ the integrity constraint:

$$\begin{aligned} r_0 :\ \leftarrow\ &bird(tweety),\, bird(polly),\, penguin(polly),\\ &not\, penguin(tweety),\, not\, flies(tweety),\, not\, flies(polly) \end{aligned}$$

is constructed.

Next we apply $Trans$. First, LP-literals which are irrelevant to the example L are dropped:

$$r_1 : \leftarrow bird(tweety),\, not\, penguin(tweety),\, not\, flies(tweety).$$

When $flies$ is the target predicate, shifting $flies(tweety)$ to the head produces

$$r_3 :\ flies(tweety) \leftarrow bird(tweety),\, not\, penguin(tweety)\,.$$

Finally, replacing $tweety$ by a variable x, we get

$$R_{ie} :\ flies(x) \leftarrow bird(x),\, not\, penguin(x)$$

where $P \cup \{R_{ie}\} \models_s L$ holds.

Example 6.2. Let P be the background program:

$$\begin{aligned} &flies(x) \leftarrow bird(x),\, not\, ab(x),\\ &bird(x) \leftarrow penguin(x),\\ &bird(tweety) \leftarrow,\\ &penguin(polly) \leftarrow . \end{aligned}$$

Given the negative example $L = not\, flies(polly)$, $P \models_s not\, not\, flies(polly)$ holds. Then, the set M^+ of LP-literals becomes

$$\begin{aligned} M^+ = \{ &bird(tweety),\, bird(polly),\, penguin(polly),\\ &not\, penguin(tweety),\, flies(tweety),\, flies(polly),\\ &not\, ab(tweety),\, not\, ab(polly)\,\}. \end{aligned}$$

Suppose that the following integrity constraint is constructed from M^+.

$$\begin{aligned} r_0: \leftarrow\ & bird(tweety),\ bird(polly),\ penguin(polly), \\ & not\, penguin(tweety),\ flies(tweety),\ flies(polly), \\ & not\, ab(tweety),\ not\, ab(polly)\,. \end{aligned}$$

By dropping LP-literals which are irrelevant to L, it becomes

$$r_1 : \leftarrow bird(polly),\ penguin(polly),\ flies(polly), not\, ab(polly)\,.$$

As $bird(polly)$ is implied by $penguin(polly)$ in P, it is dropped as

$$r_2 : \leftarrow penguin(polly),\ flies(polly), not\, ab(polly)\,.$$

Now let ab be the target predicate. Then, shifting $ab(polly)$ to the head, it becomes

$$r_3:\ ab(polly) \leftarrow penguin(polly),\ flies(polly)\,.$$

Replacing $polly$ by a variable x, we get

$$r_4:\ ab(x) \leftarrow penguin(x),\ flies(x)\,.$$

However, $P \cup \{r_4\}$ is inconsistent (i.e., having no stable model).

To get a consistent program, construct r_0' as

$$\begin{aligned} r_0':\ \leftarrow\ & bird(tweety),\ bird(polly),\ penguin(polly), \\ & not\, penguin(tweety),\ not\, ab(tweety),\ not\, ab(polly)\,. \end{aligned}$$

Applying $Trans$ to this r_0', we get

$$R_{ie}:\ ab(x) \leftarrow penguin(x)$$

where $P \cup \{R_{ie}\} \models_s L$ holds.

Our IE rule is different from Muggleton's one even in the Horn cases.

Example 6.3. [15] Let P be the background program:

$$\begin{aligned} & even(s(x)) \leftarrow odd(x), \\ & even(0) \leftarrow, \end{aligned}$$

and the positive example $L = odd(s^3(0))$ where $P \models_s not\, L$. Then, it becomes

$$M^+ = \{\, even(0),\ not\, even(s(0)),\ \ldots\ ,not\, odd(0),\ not\, odd(s(0)),\ \ldots\}.$$

Take the constraint r_0 as

$$\leftarrow even(0),\ not\, odd(s(0)).$$

By shifting $odd(s(0))$ to the head and generalizing it, the rule

$$R_{ie}:\ odd(s(x)) \leftarrow even(x)$$

is obtained. Recall that Muggleton's IE rule cannot derive this rule.[8]

[8] In a Horn logic program, M^+ has the effect which is similar to Muggleton's *enlarged bottom set* [12].

7 Discussion

The present ILP systems mostly consider Horn logic programs as background theories, and inverse entailment is used for finding hypotheses efficiently in a program. On the other hand, Horn logic programs have limitation in representing incomplete knowledge and reasoning with commonsense. This is because commonsense reasoning with incomplete knowledge is inherently nonmonotonic. This observation led to the extensive study of nonmonotonic extensions of logic programming, which includes logic programs with negation as failure [2].

Extending the representation language and enhancing reasoning ability are also indispensable for constructing a powerful ILP system. In this sense, combining induction and commonsense reasoning in the framework of *nonmonotonic ILP* is an important step in the ILP research. However, the present ILP techniques in Horn logic programs are not always applicable in their present form. For example, it is shown in [13] that inverse resolution causes some problems in the presence of NAF. Inverse entailment is also an important technique in Horn ILP, so it is meaningful to examine the technique in the nonmonotonic situation and to reconstruct a theory. There are some ILP systems which handle nonmonotonic logic programs as background theories (e.g. [1, 4, 3, 10, 7, 5]). To our best knowledge, however, no study tackles the problem of reformulating IE in nonmonotonic theories.

An inductive inference rule is *correct* if it produces a rule R satisfying $P \cup \{R\} \models_s L$ for a program P and an example L. In contrast to the IE rule in Horn logic programs, the correctness of our IE rule is susceptible to the construction of R. For one reason, in nonmonotonic logic programs a rule is viewed as a derivation rule and its representation form is meaningful. For example, given the program $P = \{\, a \leftarrow,\ \ b \leftarrow b \,\}$ and the positive example $L = c$, it becomes $M^+ = \{\, a, \mathit{not}\, b, \mathit{not}\, c \,\}$. Then, from the integrity constraint $\leftarrow a, \mathit{not}\, b, \mathit{not}\, c$, two different rules $R_1 = (c \leftarrow a, \mathit{not}\, b)$ and $R_2 = (b \leftarrow a, \mathit{not}\, c)$ are constructed according to the different choice of the target predicate. In this case, both R_1 and R_2 satisfy the relation $M^+ \models \mathit{not}\, R$, while $P \cup \{R_1\} \models_s L$ but $P \cup \{R_2\} \not\models_s L$. The failure of R_2 is due to the inappropriate selection of the target predicate. If we choose c as the target, the IE rule produces the correct rule R_1. In many cases the target predicate is the predicate in the given example, while this is not always the case as in Example 6.2.

On the other hand, the transformation sequence $Trans$ outputs rules satisfying $M^+ \models \mathit{not}\, R$, while $Trans$ also filters out useless hypotheses. For instance, in Example 6.1 the rule $\mathit{flies}(\mathit{tweety}) \leftarrow \mathit{bird}(\mathit{polly})$ explains $\mathit{flies}(\mathit{tweety})$ in P, but this rule is dropped in the process of $Trans$. In this sense, our IE rule is not necessarily *complete* for the computation of rules satisfying $P \cup \{R\} \models_s L$. However, we are dubious about the practical value of the completeness of an inductive inference rule. This is because there may exist possibly infinite solutions for explaining given examples in general, and it seems meaningless to guarantee the completeness for computing tons of useless hypotheses. What is important is selecting meaningful hypotheses in the process of computation, and $Trans$ realizes this kind of filtering. We showed in Example 6.3 that a meaningful hy-

pothesis, which is not computed by Muggleton's IE rule, is obtained using our IE rule.

Finally, we remark the computational complexity of our IE rule in the propositional case. Given a finite propositional normal logic program P, checking whether an interpretation M of P is a stable model is done in polynomial time.[9] Thus, checking whether a set M^+ of LP-literals is the expansion of the stable model of a categorical program is also executed in polynomial time. Our IE rule selects a subset of M^+ and composes R_{ie}. This task requires exponential computation in general, however, the transformation $Trans$ reduces the number of candidate hypotheses, which would ease the construction of R_{ie}.

8 Summary

This paper introduced an inverse entailment rule in nonmonotonic ILP. We provided a new entailment theorem and a contrapositive theorem to construct a theory of inverse entailment. We demonstrated that the new inverse entailment rule successfully finds appropriate inductive hypotheses in normal logic programs. The proposed method extends the present technique to a syntactically and semantically richer framework, and contributes to a theory of nonmonotonic ILP.

In this paper, we considered the IE rule for categorical programs having exactly one stable model. When a program has multiple stable models, however, the proposed technique is inapplicable in its present form. It is necessary to relax the condition for further extension.

Acknowledgements

The author thanks Katsumi Inoue for recalling Shoham's result. He also thanks Akihiro Yamamoto for clarifying the problem of Muggleton's IE rule.

References

1. M. Bain and S. Muggleton. Non-monotonic learning. In: S. Muggleton (ed.) *Inductive Logic Programming*, Academic Press, pp. 145–161, 1992.
2. C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming* 19/20:73–148, 1994.
3. F. Bergadano, D. Gunetti, M. Nicosia, and G. Ruffo. Learning logic programs with negation as failure. In: L. De Raedt (ed.) *Advances in Inductive Logic Programming*, IOS Press, pp. 107–123, 1996.
4. Y. Dimopoulos and A. Kakas. Learning nonmonotonic logic programs: learning exceptions. In: *Proceedings of the 8th European Conference on Machine Learning, Lecture Notes in Artificial Intelligence* 912, Springer-Verlag, pp. 122–137, 1995.

[9] More precisely, it takes polynomial time to derive P^M and to compute the least model of P^M. Checking the equivalence between M and the least model is also done in polynomial time.

5. L. Fogel and G. Zaverucha. Normal programs and multiple predicate learning. In: *Proceedings of the 8th International Workshop on Inductive Logic Programming, Lecture Notes in Artificial Intelligence* 1446, Springer-Verlag, pp. 175–184, 1998.
6. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In: *Proceedings of the 5th International Conference and Symposium on Logic Programming*, MIT Press, pp. 1070–1080, 1988.
7. K. Inoue and Y. Kudoh. Learning extended logic programs. In: *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 176–181, 1997.
8. V. Lifschitz, L. R. Tang, and H. Turner. Nested expression in logic programs. *Annals of Mathematics and Artificial Intelligence* 25:369–389, 1999.
9. J. W. Lloyd. *Foundations of logic programming* (2nd edition). Springer-Verlag, 1987.
10. L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In: L. De Raedt (ed.) *Advances in Inductive Logic Programming*, IOS Press, pp. 219–235, 1996.
11. S. Muggleton. Inverse entailment and Progol. *New Generation Computing* 13:245–286, 1995.
12. S. Muggleton. Completing inverse entailment. In: *Proceedings of the 8th International Workshop on Inductive Logic Programming, Lecture Notes in Artificial Intelligence* 1446, Springer-Verlag, pp. 245–249, 1998.
13. C. Sakama. Some properties of inverse resolution in normal logic programs. In: *Proceedings of the 9th International Workshop on Inductive Logic Programming, Lecture Notes in Artificial Intelligence* 1634, Springer-Verlag, pp. 279–290, 1999.
14. Y. Shoham. Nonmonotonic logics: meaning and utility. In: *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, pp. 388–393, 1987.
15. A. Yamamoto. Which hypotheses can be found with inverse entailment? In: *Proceedings of the 7th International Workshop on Inductive Logic Programming, Lecture Notes in Artificial Intelligence* 1297, Springer-Verlag, pp. 296–308, 1997.