Inductive Equivalence of Logic Programs

Chiaki Sakama[1] and Katsumi Inoue[2]

[1] Department of Computer and Communication Sciences,
Wakayama University, Sakaedani, Wakayama 640-8510, Japan
sakama@sys.wakayama-u.ac.jp
[2] National Institute of Informatics,
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
ki@nii.ac.jp

Abstract. This paper studies equivalence issues in inductive logic programming. A background theory B_1 is *inductively equivalent* to another background theory B_2 if B_1 and B_2 induce the same hypotheses for any given set of examples. Inductive equivalence is useful to compare inductive capabilities among agents having different background theories. Moreover, it provides conditions for optimizing background theories through appropriate program transformations. In this paper, we consider three different classes of background theories: clausal theories, Horn logic programs, and nonmonotonic extended logic programs. We show that logical equivalence is the necessary and sufficient condition for inductive equivalence in clausal theories and Horn logic programs. In nonmonotonic extended logic programs, on the other hand, *strong equivalence* is necessary and sufficient for inductive equivalence in general. Interestingly, however, we observe that several existing induction algorithms require weaker conditions of equivalence under restricted problem settings. We also discuss connection to equivalence in abductive logic and conclude that the notion of strong equivalence is useful to characterize equivalence of non-deductive reasoning.

1 Introduction

The issue of equivalence between logic programs is receiving increasing attention. In knowledge representation, a logic program is used for representing knowledge of a problem domain. The same problem may be encoded in different manners by different experts. Equivalence of two programs is then useful to identify different knowledge bases. In program development, one program may give a declarative specification of some problem and another program may give an efficient coding of it. In this case, equivalence of two programs guarantees a correct implementation of the given specification. Various criteria for program equivalence are proposed in the literature [5,10,13,14,15,25]. Of these, *weak equivalence* and *strong equivalence* of two programs are widely studied. Two logic programs P_1 and P_2 are *weakly equivalent* if they have the same declarative meaning. On the other hand, P_1 and P_2 are *strongly equivalent* if for any logic program R, $P_1 \cup R$ and $P_2 \cup R$ have the same declarative meaning. By the definition, strong equivalence implies weak equivalence.

S. Kramer and B. Pfahringer (Eds.): ILP 2005, LNAI 3625, pp. 312–329, 2005.

Equivalence relations presented above are intended to compare capabilities of deductive reasoning between programs. When we consider realizing intelligent agents that can perform commonsense reasoning, however, comparing capabilities of *non-deductive* reasoning between programs is also necessary and important. Recently, Inoue and Sakama argue equivalence in *abductive logic* [11]. They introduce two different types of abductive equivalence: *explainable equivalence* and *explanatory equivalence*. The former considers whether two theories have the same explainability for any observation, while the latter considers whether two theories have the same explanations for any observation. These two notions compare capabilities of abductive reasoning among agents, and [11] provides necessary and sufficient conditions for abductive equivalence in first-order logic and *abductive logic programming* [4]. *Induction* is also known as non-deductive reasoning, which is often distinguished from abduction [6]. In computational logic, induction is realized by *inductive logic programming* (ILP) [19,21]. A typical ILP problem is to induce new rules which explain given examples together with a background theory. There are several parameters which should be considered in defining equivalence notions in ILP. Several questions then arise, for instance: When can we say that induction with a background theory is equivalent to induction with another background theory? When can we say that induction from a set of examples is equivalent to induction from another set of examples? When can we say that induced hypotheses are equivalent to another induced hypotheses? Do conditions for these equivalence depend on underlying logics? These equivalence issues are important and meaningful for comparing different induction tasks, but no study answers these questions as far as the authors know.

This paper focuses on the first question presented above. A background theory B_1 is said *inductively equivalent* to another background theory B_2 if B_1 and B_2 induce the same hypothesis H (under the same hypothesis language) in face of an arbitrary set E of examples. Intuitively, if an agent has a background theory B_1 that is inductively equivalent to another background theory B_2 of another agent, then these two agents are considered equivalent with respect to inductive capability. In this case, we can identify those two agents as far as induction is concerned. On the other hand, if a theory B_1 is transformed to another syntactically different B_2, inductive equivalence of two theories guarantees identification of results of induction from each theory. This provides guidelines for optimizing background theories in ILP. The problem of interest is syntactic/semantic conditions for inductive equivalence in ILP. Conditions for inductive equivalence are arguable in different logics of background theories. In this paper, we consider three different classes of background theories – clausal theories, Horn logic programs, and nonmonotonic extended logic programs. We show that logical equivalence is the necessary and sufficient condition for inductive equivalence in clausal theories and Horn logic programs. In nonmonotonic extended logic programs, on the other hand, strong equivalence is necessary and sufficient for inductive equivalence in general. Interestingly, however, we observe that several induction algorithms require weaker conditions of equivalence between programs under restricted problem settings. We also discuss connection to equivalence in

abductive logic, and conclude that the notion of strong equivalence is useful to characterize equivalence of non-deductive reasoning.

The rest of this paper is organized as follows. Section 2 introduces the notion of inductive equivalence. Section 3 and Section 4 present inductive equivalence in clausal theories and Horn logic programs, respectively. Section 5 provides results in nonmonotonic extended logic programs. Section 6 discusses related issues and Section 7 summarizes the paper.

2 Inductive Equivalence

In this paper, we consider logical theories whose domain is given as the *Herbrand universe* and interpretations/models are defined as subsets of the *Herbrand base* HB. Given a logical theory B, let $Mod(B)$ be the set of all (Herbrand) models of B, and $SEM(B)$ the set of all *canonical* models of B. Canonical models are models that are selected from $Mod(B)$ based on some preference criterion, and the relation $SEM(B) \subseteq Mod(B)$ holds. Let L be a logic of a theory B whose semantics is given by $SEM(B)$. Then, a theory B *entails* a formula F under L (written as $B \models_L F$) if F is true in any $I \in SEM(B)$. B entails a set G of formulas under L (written as $B \models_L G$) if B entails every formula in G under L.

Remark: The meaning of the entailment relation $\models_L$ depends on underlying logic L. In this paper, different entailment relations are considered based on different logic L. As a special case, we use the reserved symbol $\models$ for logical entailment in first-order logic.

The induction problem considered in this paper is described as follows:[1]

Given: a *background theory* B, and a set E of *examples*;

Find: a *hypothesis* H such that $B \cup H$ is consistent and

$$B \cup H \models_L E\,. \tag{1}$$

When H satisfies the relation (1), we say that a hypothesis H *explains* E with respect to B (under L). Throughout the paper, a background theory B is assumed to be consistent. The examples E are *positive* examples and we do not consider negative examples in this paper.

When two different theories B_1 and B_2 are compared, they are assumed to have the common underlying (hypothesis) language. In logic programming, there are different notions of equivalence between theories. In this paper, we consider three different types of equivalence relations. Two theories B_1 and B_2 are:

- *logically equivalent* (written as $B_1 \equiv B_2$) if $Mod(B_1) = Mod(B_2)$.
- *weakly equivalent* (written as $B_1 \equiv_w B_2$) if $SEM(B_1) = SEM(B_2)$.

[1] This type of induction is called *explanatory induction*. An alternative type is considered in Section 3.2.

– *strongly equivalent* (written as $B_1 \equiv_s B_2$) if $B_1 \cup Q \equiv_w B_2 \cup Q$ for any theory Q under the same language.

By the definition, $B_1 \equiv_s B_2$ implies $B_1 \equiv_w B_2$. In particular, when $SEM(B) = Mod(B)$ holds in first-order logic, three equivalence relations coincide [5]. As canonical models, *minimal models* are often considered. The set of all minimal models of B (denoted by $MM(B)$) is defined as $MM(B) = \{ M \in Mod(B) \mid \neg\exists N \in Mod(B) \text{ s.t. } N \subset M \}$. We first show that logical equivalence coincides with strong equivalence when $SEM(B) = MM(B)$ in first-order logic. In what follows, $M^* = M \cup \{ \neg A \mid A \in HB \setminus M \}$. Then, any $M (\subseteq HB)$ is a model of B if $B \cup M^*$ is satisfiable.

Proposition 2.1. *For any first-order theories B_1 and B_2, $B_1 \equiv B_2$ iff $MM(B_1 \cup Q) = MM(B_2 \cup Q)$ for any first-order theory Q.*

Proof. The only-if part is obvious. Let $MM(B_1 \cup Q) = MM(B_2 \cup Q)$ for any Q. If $B_1 \not\equiv B_2$, there is $M \in Mod(B_1) \setminus Mod(B_2)$. Since M is not a model of B_2, $B_2 \cup M^*$ is unsatisfiable. Thus, $MM(B_2 \cup M^*) = \emptyset$. On the other hand, $M \in Mod(B_1 \cup M^*)$, so $MM(B_1 \cup M^*) \neq \emptyset$. This contradicts the assumption. Hence, $B_1 \equiv B_2$. □

By contrast, logical equivalence does not coincide with weak equivalence when $SEM(B) = MM(B)$.

Example 2.1. Consider two propositional theories:

$$B_1 : \quad a \vee b, \quad c \vee \neg a, \quad c \vee \neg b,$$
$$B_2 : \quad a \vee b, \quad c.$$

If we set $SEM(B_i) = Mod(B_i)$ for $i = 1, 2$, then $Mod(B_1) = Mod(B_2) = \{\{a, c\}, \{b, c\}, \{a, b, c\}\}$. Hence, $B_1 \equiv B_2$ and $B_1 \equiv_s B_2$. On the other hand, consider

$$B_3 : \quad a \vee b, \quad \neg a \vee \neg b, \quad c.$$

Then, $B_1 \not\equiv B_3$ and $B_2 \not\equiv B_3$. If we set $SEM(B_i) = MM(B_i)$ for $i = 1, 2, 3$, then $MM(B_1) = MM(B_3)$ and $MM(B_2) = MM(B_3)$. Hence, $B_1 \equiv_w B_3$ and $B_2 \equiv_w B_3$. By contrast, $B_1 \not\equiv_s B_3$ nor $B_2 \not\equiv_s B_3$ because the addition of $Q = \{a, b\}$ makes B_3 inconsistent.

The next definition provides a general framework of inductive equivalence between two theories.

Definition 2.1. Two theories B_1 and B_2 are *inductively equivalent* under a logic L if it holds that $B_1 \cup H \models_L E$ iff $B_2 \cup H \models_L E$ for any set E of examples and for any hypothesis H such that $B_1 \cup H$ and $B_2 \cup H$ are consistent.

By the definition, inductive equivalence presents that two background theories have the same explanation power for any example. Background theories can be represented by different logics, so that conditions of inductive equivalence are argued in respective logic L. In the following sections, we provide general conditions for inductive equivalence in different logics, and argue the issue in specific ILP algorithms.

3 Clausal Theories

We start with clausal theories in general. A *clausal theory* B is a set of clauses of the form:

$$A_1 \vee \cdots \vee A_m \vee \neg A_{m+1} \vee \cdots \vee \neg A_n$$

where A_i $(1 \leq i \leq n)$ are atoms. In the context of logic programming, it is also written as

$$A_1 \vee \cdots \vee A_m \leftarrow A_{m+1}, \ldots, A_n \,. \tag{2}$$

If $m \leq 1$ for every clause (2) in B, B is a *Horn logic program*. A Horn logic program B is *definite* if $m = 1$ for every clause (2) in B. Horn logic programs are handled in detail in Section 4. A theory, a clause or an atom is *ground* if it contains no variable. A theory or a clause with variables stands for the set of its ground instances. A *propositional* theory is a finite set of ground clauses.

Given a background theory B as a clausal theory and a set E of clauses as examples, induction produces a set H of clauses as hypothesis. As usual, a set of clauses is identified with the conjunction of clauses included in the set. We first set $SEM(B) = Mod(B)$. In this case, logical equivalence of background programs is necessary and sufficient.[2]

Theorem 3.1. *Two clausal theories B_1 and B_2 are inductively equivalent under clausal logic iff $B_1 \equiv B_2$.*

Proof. B_1 and B_2 are inductively equivalent under clausal logic
iff $B_1 \cup H \models E \;\Leftrightarrow B_2 \cup H \models E$ for any set E of clauses and for any set H of clauses such that $B_1 \cup H$ and $B_2 \cup H$ are consistent
iff $B_1 \models H \rightarrow E \;\Leftrightarrow B_2 \models H \rightarrow E$ for any H and E such that $B_1 \cup H$ and $B_2 \cup H$ are consistent
iff $B_1 \equiv B_2$. □

Since logical equivalence coincides with strong/weak equivalence under the setting $SEM(B) = Mod(B)$ in clausal logic, the above result implies that strong/weak equivalence of two theories is also necessary and sufficient.

Next, we set $SEM(B) = MM(B)$ for the semantics of a clausal theory B. Such a setting is considered as the *minimal model semantics* of disjunctive logic programs [17] or *circumscription* [16]. Then, we write $B \models_{MM} C$ if a clause C is satisfied in any $I \in MM(B)$. For any set D of clauses, we write $B \models_{MM} D$ if $B \models_{MM} C$ for every clause C in D. Under the setting, the notion of inductive equivalence *under the minimal model semantics* is defined in the same manner as Definition 2.1 with the only difference that the entailment relation $\models_L$ is replaced by $\models_{MM}$. In this case, we have the next result.

Theorem 3.2. *Two clausal theories B_1 and B_2 are inductively equivalent under the minimal model semantics iff $B_1 \equiv B_2$.*

[2] This result is also obtained as a special case of explanatory equivalence of abductive frameworks by allowing any clause as a candidate hypothesis in [11, Theorem 3.6].

Proof. Suppose that B_1 and B_2 are inductively equivalent under the minimal model semantics. Then, $B_1 \cup H \models_{MM} E$ iff $B_2 \cup H \models_{MM} E$ for any set H and for any set E of clauses such that $B_1 \cup H$ and $B_2 \cup H$ are consistent. By putting $E = B_1 \cup H$, it holds that $B_1 \cup H \models_{MM} B_1 \cup H$ iff $B_2 \cup H \models_{MM} B_1 \cup H$. By putting $E = B_2 \cup H$, it holds that $B_1 \cup H \models_{MM} B_2 \cup H$ iff $B_2 \cup H \models_{MM} B_2 \cup H$. As $B_1 \cup H \models_{MM} B_1 \cup H$ and $B_2 \cup H \models_{MM} B_2 \cup H$ always hold, $B_1 \cup H \models_{MM} B_2 \cup H$ and $B_2 \cup H \models_{MM} B_1 \cup H$ also hold. By $B_1 \cup H \models_{MM} B_2 \cup H$, any minimal model M of $B_1 \cup H$ satisfies every clause in $B_2 \cup H$. If $M \notin MM(B_2 \cup H)$, there is a minimal model $N \in MM(B_2 \cup H)$ such that $N \subset M$ and N satisfies $B_2 \cup H$. By $B_2 \cup H \models_{MM} B_1 \cup H$, N satisfies every clause in $B_1 \cup H$. But this is impossible because M is a minimal model of $B_1 \cup H$. Hence, $M \in MM(B_2 \cup H)$. Likewise, $M \in MM(B_2 \cup H)$ implies $M \in MM(B_1 \cup H)$. Therefore, $MM(B_1 \cup H) = MM(B_2 \cup H)$, so that $B_1 \equiv B_2$ by Proposition 2.1.

Conversely, if $B_1 \equiv B_2$, $MM(B_1 \cup H) = MM(B_2 \cup H)$ holds for any set H of clauses (Proposition 2.1). Then, $B_1 \cup H \models_{MM} E$ iff $B_2 \cup H \models_{MM} E$ for any set E of clauses and $B_1 \cup H$ is consistent iff $B_2 \cup H$ is consistent. Hence, B_1 and B_2 are inductively equivalent. □

Theorem 3.2 and Proposition 2.1 imply that strong equivalence of two theories is also necessary and sufficient for inductive equivalence under the minimal model semantics. Note that weak equivalence of two theories is not sufficient for inductive equivalence.

Example 3.1. Two theories $B_1 = \{\, p(x) \vee \neg q(x), \quad r(a) \,\}$ and $B_2 = \{\, r(a) \,\}$ have the same minimal model $\{r(a)\}$, thereby weakly equivalent. However, they are not inductively equivalent. In fact, for the example $E = \{p(a)\}$, the clause $H = (q(x) \vee \neg r(x))$ explains $p(a)$ in B_1, but not in B_2.

Theorems 3.1 and 3.2 imply that in full clausal theories the notion of inductive equivalence under $SEM(B) = Mod(B)$ and the one under $SEM(B) = MM(B)$ coincide.

Corollary 3.3. *Two clausal theories are inductively equivalent under clausal logic iff they are inductively equivalent under the minimal model semantics.*

Given two propositional clausal theories B_1 and B_2, the problem of testing $B_1 \equiv B_2$ is equivalent to the problem of testing unsatisfiability of $B_1 \wedge \neg B_2$, which is coNP-complete. Then, the next result follows by Theorems 3.1 and 3.2.

Corollary 3.4. *Deciding inductive equivalence of two propositional clausal theories is coNP-complete.*

In what follows, we pick up two induction methods for full clausal theories and investigate conditions for inductive equivalence.

3.1 CF-Induction

Inoue [9] provides a method for induction from full clausal theories. It is based on the technique of *consequence finding* (CF). Given a background theory B as

a clausal theory and a set E of examples as clauses, *CF-induction* computes a hypothesis H as follows: First, the condition $B \cup H \models E$ of (1) is converted to

$$B \cup \{\neg E\} \models \neg H$$

where $\neg E$ is a formula in a disjunctive normal form. The above relation is interpreted as $B \cup \{\neg E\} \models CC(B, E)$ and $CC(B, E) \models \neg H$ with some clausal theory $CC(B, E)$. The relation $CC(B, E) \models \neg H$ is then rewritten as

$$H \models \neg CC(B, E)\,.$$

Then, a hypothesis H is constructed as a clausal theory which entails $\neg CC(B, E)$.[3] $CC(B, E)$ is a set of clauses that are computed by the *characteristic clauses* of $B \cup \{\neg E\}$. The characteristic clauses of a set Σ of clauses are defined as follows. A clause C *subsumes* a clause D if $C\theta \subseteq D$ for some substitution θ. C *properly subsumes* D if C subsumes D but D does not subsume C. Then,

$$Carc(\Sigma) = \{\, C \in Th(\Sigma) \mid \neg\exists\, D \in Th(\Sigma) \text{ s.t. } D \text{ properly subsumes } C \,\}.$$

That is, each characteristic clause is a theorem of Σ that is not properly subsumed by any clause in the set of theorems. Then, it holds that

$$Carc(B \cup \{\neg E\}) \models CC(B, E).$$

Inductive equivalence under CF-induction is then defined as follows.

Definition 3.1. Let B_1 and B_2 are two clausal theories. Then, B_1 and B_2 are *inductively equivalent under CF-induction* if $Carc(B_1 \cup \{\neg E\}) = Carc(B_2 \cup \{\neg E\})$ for any set E of clauses.

Then, we have the next result.

Theorem 3.5. *Let B_1 and B_2 be two clausal theories. Then, B_1 and B_2 are inductively equivalent under CF-induction iff $B_1 \equiv B_2$.*

Proof. If $Carc(B_1 \cup \{\neg E\}) = Carc(B_2 \cup \{\neg E\})$ for any set E of clauses, $Carc(B_1) = Carc(B_2)$ by putting $E = \emptyset$. This implies $B_1 \equiv B_2$. The converse is straightforward. □

The above result, together with Theorem 3.1, implies that B_1 and B_2 are inductively equivalent under clausal logic iff they are inductively equivalent under CF-induction.

3.2 Confirmatory Induction

Confirmatory induction (or *descriptive induction*) is an alternative framework of induction [2]. In this framework, a hypothesis H explains an example E with respect to a background theory B iff H is satisfied by every $I \in SEM(B \cup E)$.

[3] This extends Muggleton's *inverse entailment* in Horn theories. Muggleton's method is explained in Section 4.2.

The system CLAUDIEN [3] realizes this type of induction under the minimal model semantics $SEM = MM$.

The notion of inductive equivalence in this context is distinguished as *confirmatorily inductive equivalence* (*c-inductive equivalence*, for short). The notion of c-inductive equivalence is defined as follows.

Definition 3.2. Two theories B_1 and B_2 are *c-inductively equivalent* under a logic L if it holds that $B_1 \cup E \models_L H$ iff $B_2 \cup E \models_L H$ for any set E of examples and for any hypothesis H such that $B_1 \cup H$ and $B_2 \cup H$ are consistent.

When a background theory is given as a clausal theory, c-inductive equivalence under the minimal model semantics is characterized as follows.

Theorem 3.6. *Two clausal theories B_1 and B_2 are c-inductively equivalent under the minimal model semantics iff $B_1 \equiv B_2$.*

Proof. Suppose that B_1 and B_2 are c-inductively equivalent under the minimal model semantics. Then, $B_1 \cup E \models_{MM} H$ iff $B_2 \cup E \models_{MM} H$ for any set E and for any set H of clauses such that $B_1 \cup H$ and $B_2 \cup H$ are consistent. Then, $B_1 \equiv B_2$ holds by Theorem 3.2. The converse is shown in a straightforward manner. □

The above theorem presents that in clausal theories two notions of inductive equivalence coincide (under the minimal model semantics). That is, B_1 and B_2 are inductively equivalent in explanatory induction iff they are inductively equivalent in confirmatory induction.

4 Induction in Horn Logic Programs

Next we consider the case where a background theory, examples, and hypotheses are all Horn logic programs. A lot of ILP systems handle Horn logic programs and some algorithms are known for Horn ILP. So we discuss here inductive equivalence in Horn logic programs apart from Section 3.

The declarative semantics of a Horn logic program is given by the unique minimal model, called the *least model*. Thus, for any Horn logic program B, $SEM(B) = MM(B)$ and we write $MM(B)$ as $LM(B)$. We write $B \models_{LM} C$ if a Horn clause C is satisfied in $LM(B)$. For a set D of Horn clauses, we write $B \models_{LM} D$ if $B \models_{LM} C$ for every Horn clause C in D. Notice that $\models_{LM}$ does not coincide with $\models$. For instance, given $B = \{p \leftarrow q\}$, $B \models_{LM} \neg p$ but $B \not\models \neg p$. Thus, $\models_{LM}$ has the effect of the *closed world assumption*.

Definition 4.1. Two Horn logic programs B_1 and B_2 are *inductively equivalent (under the least model semantics)* if it holds that $B_1 \cup H \models_{LM} E$ iff $B_2 \cup H \models_{LM} E$ for any set E of examples and for any hypothesis H such that $B_1 \cup H$ and $B_2 \cup H$ are consistent.

For inductive equivalence in Horn logic programs, the next result follows by Theorem 3.2.

Theorem 4.1. *Let B_1 and B_2 be two Horn logic programs. Then, B_1 and B_2 are inductively equivalent iff $B_1 \equiv B_2$.*

Logical equivalence of two propositional Horn logic programs is tested in polynomial-time, so that:

Proposition 4.2. *Deciding inductive equivalence of two propositional Horn logic programs is done in polynomial-time.*

Several algorithms are known for induction in Horn logic programs. We investigate conditions for inductive equivalence in two popular algorithms.

4.1 Relative Least General Generalization

Plotkin's *relative least general generalization* [24] is a well-known algorithm for induction, which is used in the Horn ILP system GOLEM [18]. We first remind terms and basic results. A clause C_1 *subsumes* another clause C_2 *relative to* a program B, denoted by $C_1 \succeq_B C_2$, if there is a substitution θ such that $B \models C_1\theta \rightarrow C_2$. A clause D is a *relative least general generalization* (RLGG) of C_1 and C_2 with respect to B if D is the least upper bound of C_1 and C_2 under the ordering $\succeq_B$ over the clausal language. The RLGG does not always exist but exists when B is a ground program.

Inductive equivalence under RLGG is defined as follows. Given a ground Horn logic program B and a set E of ground Horn clauses, let $RLGG(B, E)$ be the set of clauses which are the RLGG of clauses in E with respect to B.

Definition 4.2. Let B_1 and B_2 be two ground Horn logic programs. Then, B_1 and B_2 are *inductively equivalent under RLGG* if $RLGG(B_1, E) = RLGG(B_2, E)$ for any set E of ground Horn clauses.

Given a ground Horn logic program B and examples E as a set of ground Horn clauses, GOLEM constructs inductive hypothesis H as follows:

$$\begin{aligned} & B \cup H \models E \\ \Leftrightarrow \quad & H \models B \rightarrow E \\ \Leftrightarrow \quad & \models H \rightarrow (\neg B \vee E). \end{aligned}$$

At this point, GOLEM replaces B with the conjunction of ground atoms included in a finite subset of $LM(B)$. Here we suppose that the $LM(B)$ is finite. Then, we replace B with $LM(B)$ as GOLEM does. Let $E = \{C_1, \ldots, C_k\}$. Then, the RLGG of E with respect to B is computed as the least general generalization (LGG) of clauses $(C_1 \vee \neg LM(B)), \ldots, (C_k \vee \neg LM(B))$, where $\neg LM(B) = \bigvee_{A_i \in LM(B)} \neg A_i$, which becomes a solution H. Then, we have the following result.[4]

Theorem 4.3. *Let B_1 and B_2 be two ground Horn logic programs. Then, B_1 and B_2 are inductively equivalent under RLGG iff $B_1 \equiv_w B_2$.*

[4] We assume the result in the context of GOLEM.

Proof. Suppose that B_1 and B_2 are inductively equivalent under RLGG. Then, for any set $E = \{ C_1, \ldots, C_k \}$ of ground clauses, $RLGG(B_1, E) = RLGG(B_2, E)$ implies $LGG(C_1 \vee \neg LM(B_1), \ldots, C_k \vee \neg LM(B_1)) = LGG(C_1 \vee \neg LM(B_2), \ldots, C_k \vee \neg LM(B_2))$. Put $E = \{A\}$ for any ground atom A. Then, $LGG(A \vee \neg LM(B_1)) = A \vee \neg LM(B_1)$ and $LGG(A \vee \neg LM(B_2)) = A \vee \neg LM(B_2)$, so $LGG(A \vee \neg LM(B_1)) = LGG(A \vee \neg LM(B_2))$ implies $LM(B_1) = LM(B_2)$. Hence, $B_1 \equiv_w B_2$.

Conversely, if $B_1 \equiv_w B_2$, $LM(B_1) = LM(B_2)$. Then, for any set $E = \{ C_1, \ldots, C_k \}$ of ground clauses, $LGG(C_1 \vee \neg LM(B_1), \ldots, C_k \vee \neg LM(B_1)) = LGG(C_1 \vee \neg LM(B_2), \ldots, C_k \vee \neg LM(B_2))$, so $RLGG(B_1, E) = RLGG(B_2, E)$. Hence, the result holds. □

Example 4.1. Consider two programs:

$$\begin{aligned} B_1 :\ & has\_wings(joe) \leftarrow bird(joe), \\ & bird(tweety) \leftarrow, \\ & bird(polly) \leftarrow . \\ B_2 :\ & bird(tweety) \leftarrow, \\ & bird(polly) \leftarrow . \end{aligned}$$

Given the example $E = \{ flies(tweety), flies(polly) \}$, both the RLGG of E wrt B_1 and the RLGG of E wrt B_2 become

$$flies(x) \leftarrow bird(x) .$$

This means that the first clause in B_1 is of no use for induction under *RLGG*. Note that B_1 and B_2 are not strongly equivalent.

4.2 Inverse Entailment

Next, we consider Muggleton's *inverse entailment* (IE) algorithm which is used in the Horn ILP system PROGOL [20]. Given a Horn logic program B and a ground Horn clause E as an example, suppose a Horn clause H satisfying

$$B \cup \{H\} \models E.$$

By inverting the entailment relation it becomes

$$B \cup \{\neg E\} \models \neg H.$$

Put $\neg Bot(B, E)$ as the conjunction of ground literals which are true in every model of $B \cup \{\neg E\}$. Then, a clause H is induced by *inverse entailment* (IE) if $H \models Bot(B, E)$.

Remark: The process of inverting entailment is similar to CF-induction in clausal theories [9], but IE supposes a Horn logic program B, and a single Horn clause H and E. Another difference is that CF-induction is sound and complete for finding hypotheses, but IE is sound but not complete in general [27].

Given a Horn logic program B and a ground Horn clause E, let $IE(B, E)$ be the set of clauses which is induced by IE from E with respect to B. Then, inductive equivalence under IE is defined as follows.

Definition 4.3. Two Horn logic programs B_1 and B_2 are *inductively equivalent under IE* if $IE(B_1, E) = IE(B_2, E)$ for any ground Horn clause E.

Then, we have the following result.

Theorem 4.4. *Two Horn logic programs B_1 and B_2 are inductively equivalent under IE iff $B_1 \equiv B_2$.*

Proof. B_1 and B_2 are inductively equivalent under IE iff $Bot(B_1, E) = Bot(B_2, E)$ for any E. Then, $B_1 \cup \{\neg E\} \models L$ iff $B_2 \cup \{\neg E\} \models L$ for any ground Horn clause E and for any literal L. Put $E = A_0 \leftarrow A_1, \ldots, A_n$. Then, $B_1 \cup \{\neg A_0, A_1, \ldots, A_n\} \models L$ iff $B_2 \cup \{\neg A_0, A_1, \ldots, A_n\} \models L$ for any $\{\neg A_0, A_1, \ldots, A_n\}$. Thus, for any set F of ground atoms $B_1 \cup F$ and $B_2 \cup F$ have the same least model. Hence, $B_1 \equiv_s B_2$, thereby $B_1 \equiv B_2$ (by Proposition 2.1). The converse is straightforward. □

The results of Sections 4.1 and 4.2 show that inductive equivalence under RLGG requires a weaker condition of equivalence than IE.

5 Induction in Nonmonotonic Logic Programs

Nonmonotonic logic programs are logic programs with *negation as failure* [1]. We consider the class of *extended logic programs* [8] in this paper. An extended logic program (ELP) is a set of *rules* of the form:

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \quad (n \geq m) \tag{3}$$

where each L_i is a literal and *not* represents *negation as failure* (NAF). The literal L_0 is the *head* and the conjunction $L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$ is the *body*. A rule with the empty head of the form:

$$\leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \quad (n \geq 1) \tag{4}$$

is an *integrity constraint*. A rule with the empty body $L \leftarrow$ is a *fact* and identified with the literal L. An ELP is called a *normal logic program* (NLP) if every literal appearing in the program is an atom. Let Lit be the set of all ground literals in the language of a program. Any element in $Lit^+ = Lit \cup \{ not\ L \mid L \in Lit \}$ is called an *LP-literal* and an LP-literal $not\ L$ is called an *NAF-literal*. A rule is *NAF-free* if it contains no NAF-literal. A program is NAF-free if it consists of NAF-free rules. A program, a rule or an LP-literal is *ground* if it contains no variable. A program or a rule with variables stands for the set of its ground instances. A *propositional* program is a finite set of ground rules.

Remark: A primary difference between nonmonotonic logic programs and clausal theories is that a rule (3) is *not* a clause even if it is NAF-free. For instance, a rule $L_1 \leftarrow L_2$ has meaning different from $\neg L_2 \leftarrow \neg L_1$ or $L_1 \vee \neg L_2$.

A set $S(\subset Lit)$ *satisfies* a ground rule R of the form (3) if $\{L_1, \ldots, L_m\} \subseteq S$ and $\{ L_{m+1}, \ldots, L_n \} \cap S = \emptyset$ imply $L_0 \in S$. In particular, S satisfies a ground integrity constraint of the form (4) if $\{L_1, \ldots, L_m\} \not\subseteq S$ or $\{ L_{m+1}, \ldots, L_n \} \cap S \neq \emptyset$. When a rule R contains variables, S satisfies R if S satisfies every ground instance of R. The semantics of ELPs is given by the *answer set semantics* [8]. First, let B be an NAF-free program and $S \subset Lit$. Then, S is an *answer set* of B if (i) S is a minimal set which satisfies every ground rule in the ground instantiation of B, and (ii) S does not contain both L and $\neg L$ for any $L \in Lit$. Next, let B be any ELP and $S \subset Lit$. Then, define the NAF-free program B^S as follows: a rule $L_0 \leftarrow L_1, \ldots, L_m$ is in B^S iff there is a ground rule $L_0 \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n$ in the ground instantiation of B such that $\{ L_{m+1}, \ldots, L_n \} \cap S = \emptyset$. Here, L_0 is possibly empty. Then, S is an *answer set* of B if S is an answer set of B^S. In NLPs, answer sets coincide with *stable models* [7]. An ELP may have none, one, or multiple answer sets. The set of all answer sets of B is denoted by $AS(B)$. An ELP B is *consistent* if it has an answer set; otherwise B is *inconsistent*. An ELP B is called *categorical* if it has the unique answer set [1]. If a ground rule R is satisfied in every answer set of B, it is written as $B \models_{AS} R$. In particular, $B \models_{AS} L$ if a ground literal L is included in every answer set of B. For a set E of ground rules/literals, we write $B \models_{AS} E$ if $B \models_{AS} R$ for any $R \in E$.

An induction problem considered in this section is stated as follows. Given a consistent ELP B as a background theory and a set E of rules as examples, find a set H of rules such that $B \cup H$ is consistent and

$$B \cup H \models_{AS} E. \tag{5}$$

We put $SEM(B) = AS(B)$ in this section.

Definition 5.1. Two ELPs B_1 and B_2 are *inductively equivalent (under the answer set semantics)* if it holds that $B_1 \cup H \models_{AS} E$ iff $B_2 \cup H \models_{AS} E$ for any set E of examples and for any hypothesis H such that $B_1 \cup H$ and $B_2 \cup H$ are consistent.

We proceed to build conditions for inductive equivalence in ELPs. In what follows, we assume the underlying language of programs is function-free and Lit is finite. In this setting, every answer set is a finite set of ground literals.

Theorem 5.1. *Let B_1 and B_2 be two ELPs. Then, B_1 and B_2 are inductively equivalent iff $B_1 \equiv_s B_2$.*

Proof. When B_1 and B_2 are inductively equivalent, it holds that $B_1 \cup H \models_{AS} E$ iff $B_2 \cup H \models_{AS} E$ for any set E of rules and any set H of rules such that $B_1 \cup H$ and $B_2 \cup H$ are consistent. Suppose that there is a set S such that $S \in AS(B_1 \cup H) \setminus AS(B_2 \cup H)$ for some H. Let $S = \{L_1, \ldots, L_m\}$ and $Lit \setminus S = \{L_{m+1}, \ldots, L_n\}$. Put $E = \{ \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \}$. Then, every answer set of $B_2 \cup H$ satisfies E, but S does not satisfy E. This contradicts the assumption. Thus, no such S exists for any H and $AS(B_1 \cup H) = AS(B_2 \cup H)$. Hence, $B_1 \equiv_s B_2$. The converse is proved in a straightforward manner. □

The complexity of testing strong equivalence of two propositional ELPs is coNP-complete [14]. This implies the next result.

Proposition 5.2. *Deciding inductive equivalence of two propositional ELPs is coNP-complete.*

5.1 Induction from Answer Sets

Sakama [26] introduces an algorithm called *induction from answer sets (IAS)*. He provides procedures for handling positive/negative examples, and we review the procedure for positive examples here.

Some notions are defined. For any LP-literal L, $pred(L)$ denotes the predicate of L and $const(L)$ denotes the set of constants appearing in L. A rule (3) is *negative-cycle-free* if $pred(L_0) \neq pred(L_i)$ for any $i = m+1, \ldots, n$. Let L be a ground LP-literal and S a set of ground LP-literals. Then, L_1 in S is *relevant* to L if either (i) $pred(L_1) = pred(L)$ and $const(L_1) = const(L)$, or (ii) for some LP-literal L_2 in S, $const(L_1) \cap const(L_2) \neq \emptyset$ and L_2 is relevant to L. A ground NAF-literal $not\, L$ is *involved* in B if L appears in the ground instance of B. For simplicity reasons, the following conditions are assumed; a function-free and categorical ELP B as a background program; and a positive example as a ground literal L such that $B \not\models_{AS} L$ and $pred(L)$ appears nowhere in B.

Suppose that B has the answer set S. Then, construct a rule $L \leftarrow \Gamma$ where $\Gamma \subseteq S \cup \{not\, L \mid L \in Lit \setminus S\}$ and every element in Γ is relevant to L and is involved in B. Next, the rule $L \leftarrow \Gamma$ is generalized to R as $R\theta = (L \leftarrow \Gamma)$ with some substitution θ.

Example 5.1. ([26]) Suppose the background program B

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

which has the answer set

$$S = \{\, bird(tweety),\, bird(polly),\, penguin(polly)\}.$$

Given the example $L = flies(tweety)$, the rule $L \leftarrow \Gamma$ becomes

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

Replacing $tweety$ by a variable x, the rule

$$R:\ flies(x) \leftarrow bird(x),\, not\ penguin(x)$$

becomes a solution.

It is shown in [26] that the rule $H = \{R\}$ satisfies the condition (5) for $E = \{L\}$ if R is negative-cycle-free.

Let $IAS(B, L)$ be the set of rules which is computed by the above procedure using B and L. Then, inductive equivalence under IAS is defined as follows.

Definition 5.2. Two function-free categorical ELPs B_1 and B_2 are *inductively equivalent under IAS* if $IAS(B_1, L) = IAS(B_2, L)$ for any ground literal L.

The necessary and sufficient condition for inductive equivalence under IAS is as follows.

Theorem 5.3. *Two function-free categorical ELPs B_1 and B_2 are inductively equivalent under IAS iff $B_1 \equiv_w B_2$.*

Proof. Since the rule R is constructed by the answer set of a program, the result immediately follows. □

5.2 Induction of Stable Models

Otero [22] characterizes induction problems in normal logic programs (NLPs) under the stable model semantics. He introduces different types of induction for positive/negative examples, but here we consider the so-called *induction from non-complete sets* which is the usual ILP setting for positive examples.

Suppose a background program B as an NLP, and a set E of ground atoms as positive examples such that $B \not\models_{AS} E$.[5] The goal is to find a set H of rules satisfying the relation (5). An interpretation M is a *monotonic model* of an NLP if M satisfies every rule in B. Given a set E of examples, an interpretation M (of $B \cup E$) is an *extension* of E iff $E \subseteq M$. He then captures H satisfying (5) as an extension M of E that becomes a stable model of $B \cup M$. Note that in this definition a hypothesis H is given as a set of ground atoms.

Let $ISM(B, E)$ be the collection of H defined as above. Then, inductive equivalence under ISM is defined as follows.

Definition 5.3. Two NLPs B_1 and B_2 are *inductively equivalent under ISM* if $ISM(B_1, E) = ISM(B_2, E)$ for any set E of ground atoms.

Lemma 5.4. *[22] Given an NLP B, M is a monotonic model of B iff M is a stable model of $B \cup M$.*

Let $MonMod(B)$ be the set of monotonic models of B. Then, we have the following result.

Theorem 5.5. *Two NLPs B_1 and B_2 are inductively equivalent under ISM iff $MonMod(B_1) = MonMod(B_2)$.*

Proof. Suppose that B_1 and B_2 are inductively equivalent under ISM. For any $M \in ISM(B_1, E)$, M is a stable model of $B_1 \cup M$ and a monotonic model of B_1 (Lemma 5.4). Then, $ISM(B_1, E) = ISM(B_2, E)$ implies $MonMod(B_1) = MonMod(B_2)$. The converse is proved in a similar manner. □

[5] Recall that answer sets coincide with stable models in NLPs.

Example 5.2. Let $B_1 = \{ p \leftarrow not\, q \}$ and $B_2 = \{ q \leftarrow not\, p \}$. For $E = \{p\}$, put its extension as $M = \{p\}$. Then, $H = \{ p \leftarrow \}$ becomes a solution in both B_1 and B_2. Note that $B_1 \not\equiv_w B_2$ but $MonMod(B_1) = MonMod(B_2)$.

6 Discussion

Equivalence of logic programs has been studied in various aspects, but to our best knowledge, equivalence issue in inductive logic programming has never been discussed. Recently, Inoue and Sakama study equivalence of abductive frameworks [11]. They introduce two different types of abductive equivalence: *explainable equivalence* and *explanatory equivalence*. Given a background theory B and a set H of candidate hypotheses (called *abducibles*), an abductive framework is defined as a tuple $\langle B, H \rangle$. Two abductive frameworks $\langle B_1, H_1 \rangle$ and $\langle B_2, H_2 \rangle$ are called *explainable equivalent* if, for any observation O, there is an explanation of O in $\langle B_1, H_1 \rangle$ iff there is an explanation of O in $\langle B_2, H_2 \rangle$. On the other hand, two programs are called *explanatorily equivalent* if, for any observation O, E is an explanation of O in $\langle B_1, H_1 \rangle$ iff E is an explanation of O in $\langle B_2, H_2 \rangle$. Explanatory equivalence is stronger than explainable equivalence, and the former implies the latter.

Comparing [11] with our present work, some interesting connections are observed. When underlying logic is first-order logic, logical equivalence of two theories is a necessary and sufficient condition for explanatory equivalence. When a background theory is represented by a (nonmonotonic) logic program, on the other hand, $\langle B_1, H \rangle$ and $\langle B_2, H \rangle$ are explanatorily equivalent iff B_1 and B_2 are strongly equivalent. Those results have connection to the results of Theorems 3.1, 3.2, 4.1 and 5.1 of this paper. In particular, in clausal logic the notion of inductive equivalence coincides with the notion of explanatory equivalence if one permits arbitrary clauses as abducibles. However, there is an important difference between explanatory equivalence in abductive frameworks and inductive equivalence in this paper, which stems from the difference between abduction and induction. In an abductive framework, a hypothesis space is prespecified as H and possible explanations for a given observation are constructed as a subset of abducibles. The existence of H in abductive logic programs results in characterization by *relative* strong equivalence, i.e., B_1 and B_2 are explanatory equivalent iff they are strongly equivalent *with respect to H*. Moreover, in abductive logic programming, abducibles and observations are usually restricted to (ground) literals. In ILP, on the other hand, hypotheses and examples are general rules rather than facts. Besides these differences, both abduction and induction require strong equivalence of two (nonmonotonic) logic programs to identify the results of abductive/inductive inference. The essence of this lies in the fact that abduction and induction are both *ampliative* reasoning and extend theories. Strong equivalence takes the influence of addition of a rule set to each program into account, so that it succeeds in characterizing the effect of abduction/induction that are not captured by weak equivalence of programs. In [13], it is argued that strong equivalence is useful to simplify a part of a program with-

out looking at the other parts. On the other hand, the study [11] and the result of this paper reveal that strong equivalence has another important applications for testing equivalence in abductive and inductive logic programming.

From the computational viewpoint, testing strong equivalence of propositional nonmonotonic logic programs is converted to the problem of propositional entailment in classical logic [14]. The problem of testing strong equivalence is then solved using existing SAT solvers. For predicate programs without function symbols, strong equivalence testing is also possible by instantiating a program into a finite propositional one. There is a system for testing strong equivalence of function-free nonmonotonic logic programs, e.g., [12]. Existence of no procedure for testing strong equivalence of logic programs with functions would restrict practical application of inductive equivalence in ILP. Nevertheless, inductive equivalence is useful when background knowledge is given as a function-free Datalog or a database that is a collection of propositional sentences.

Apart from the general ILP setting, we have shown that several existing ILP algorithms require weaker conditions of equivalence between programs. Each algorithm is designed to work in some restricted problem setting for theoretical/practical reasons, and such restriction has the effect of relaxing conditions of inductive equivalence. Note that it may happen that some algorithm may produce different hypotheses from two background theories due to its incompleteness. Thus, if two strongly equivalent programs produce different hypotheses in face of some common examples, it indicates that the algorithm is incomplete or incorrect. Thus, inductive equivalence would be used for testing correctness/completeness of an algorithm. In this respect, inverse entailment in Horn logic programs is incomplete, but Theorem 4.4 guarantees that under the restricted problem setting the algorithm correctly judges inductive equivalence of two Horn logic programs. For another application, inductive equivalence would be used for comparing different induction algorithms under the common problem setting. Let $\alpha(B, E)$ be the set of hypotheses induced by an algorithm α using a background theory B and examples E. For two different algorithms α_1 and α_2 in the common problem setting, suppose that $\alpha_1(B_1, E) = \alpha_1(B_2, E)$ implies $\alpha_2(B_1, E) = \alpha_2(B_2, E)$, but not vice versa. In this case, α_1 is considered inductively more sensitive than α_2 in the sense that α_1 may distinguish different background theories that are not distinguished by α_2. For instance, suppose any ground Horn logic program B and any ground Horn clause E. In this problem setting, we can say that IE is inductively more sensitive than $RLGG$ by the result of Section 4. Thus, the notion of inductive equivalence is also useful to compare capabilities of different induction algorithms.

From the viewpoint of program development, it is known that some basic transformations including *unfolding/folding* do not preserve strong equivalence of logic programs [23]. This fact, together with the result of this paper, implies that such basic program transformations are *not* applicable to optimize background theories in ILP. If applied, the result of induction may change in general. Those transformations are still effective as far as one uses induction algorithms that require the condition of weak equivalence.

7 Conclusion

This paper has studied equivalence issues in inductive logic programming. We introduced the notion of inductive equivalence which compares inductive capabilities between different background theories. Three different logics are considered – clausal theories, Horn logic programs, and nonmonotonic extended logic programs. Logical equivalence is necessary and sufficient for inductive equivalence in clausal theories and Horn logic programs, while strong equivalence is necessary and sufficient in nonmonotonic extended logic programs. Under restricted problem settings, on the other hand, we also observed that several existing ILP algorithms require weaker conditions of equivalence. The results of this paper, together with those of [11], reveal that the notion of strong equivalence is useful to characterize equivalence in non-deductive reasoning.

In the introduction, we posed several questions on equivalence issues which may arise in ILP. This paper has answered one question regarding equivalence of background theories. Answering other questions is left for future study.

References

1. C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.
2. L. De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95:187–201, 1997.
3. L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.
4. M. Denecker and A. Kakas. Abductive logic programming. In: A. C. Kakas and F. Sadri (eds.), *Computational Logic: Logic Programming and Beyond—Essays in Honour of Robert A. Kowalski, Part I*, Lecture Notes in Artificial Intelligence, vol. 2407, pp. 402–436, Springer-Verlag, 2002.
5. T. Eiter and M. Fink. Uniform equivalence of logic programs under the stable model semantics. In: *Proceedings of the 19th International Conference on Logic Programming*, Lecture Notes in Computer Sciences, vol. 2916, pp. 224–238, Springer-Verlag, 2003.
6. P. A. Flach and A. C. Kakas (eds.). *Abduction and Induction — Essays on their Relation and Integration*. Kluwer Academic, 2000.
7. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In: *Proceedings of the 5th International Conference and Symposium on Logic Programming*, pp. 1070–1080, MIT Press, 1988.
8. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In: *Proceedings of the 7th International Conference on Logic Programming*, pp. 579–597, MIT Press, 1990.
9. K. Inoue. Induction as consequent finding. *Machine Learning*, 55:109–135, 2004.
10. K. Inoue and C. Sakama. Equivalence of logic programs under updates. In: *Proceedings of the 9th European Conference on Logics in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, vol. 3229, pp. 174–186, Springer-Verlag, 2004.
11. K. Inoue and C. Sakama. Equivalence in abductive logic. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, to appear, 2005.

12. T. Janhunen and E. Oikarinen. LPEQ and DLPEQ – translators for automated equivalence testing of logic programs. In: *Proceedings of the 7th International Conference of Logic Programming and Nonmonotonic Reasoning*, Lecture Notes in Artificial Intelligence, vol. 2923, pp. 336–340, Springer-Verlag, 2004.
13. V. Lifschitz, D. Pearce and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.
14. F. Lin. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In: *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 170–176, Morgan Kaufmann, 2002.
15. M. J. Maher. Equivalence of logic programs. In: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 627–658, Morgan Kaufmann, 1988.
16. J. McCarthy. Circumscription – a form of nonmonotonic reasoning. *Artificial Intelligence*, 13(1&2):27–39, 1980.
17. J. Minker. On indefinite data bases and the closed world assumption. In: *Proceedings of the 6th International Conference on Automated Deduction*, Lecture Notes in Computer Science, vol. 138, pp. 292–308, Springer-Verlag, 1982.
18. S. Muggleton and C. Feng. Efficient induction algorithm. In: [19], pp. 281–298, 1992.
19. S. Muggleton (ed.). *Inductive Logic Programming*, Academic Press, 1992.
20. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.
21. S.-H. Nienhuys-Cheng and R. De Wolf. *Foundations of inductive logic programming*. Lecture Notes in Artificial Intelligence, vol. 1228, Springer-Verlag, 1997.
22. R. P. Otero. Induction of stable models. In: *Proceedings of the 11th International Conference on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence, vol. 2157, pp. 193–205, Springer-Verlag, 2001.
23. M. Osorio, J. A. Navarro, and J. Arrazola. Equivalence in answer set programming. In: *Proceedings of the 11th International Workshop on Logic Based Program Synthesis and Transformation*, Lecture Notes in Computer Science, vol. 2372, pp. 57–75, Springer-Verlag, 2001.
24. G. D. Plotkin. A further note on inductive generalization. In: B. Meltzer and D. Michie (eds.), *Machine Intelligence*, vol. 6, pp. 101–124, Edinburgh University Press, 1971.
25. Y. Sagiv. Optimizing Datalog programs. In: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 659–668, Morgan Kaufmann, 1988.
26. C. Sakama. Induction from answer sets in nonmonotonic logic programs. *ACM Transactions on Computational Logic*, 6(2):203–231, 2005. Preliminary version: Learning by answer sets. In: *Proceedings of the AAAI Spring Symposium on Answer Set Programming*, pp. 181–187, AAAI Press, 2001.
27. 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, vol. 1297, pp. 296–308, Springer-Verlag, 1997.