ICOT Technical Report: TR-735

TR-735

A Formalization of Generalization-Based Analogy in General Logic Programs

by
N. Iwayama, K. Satoh & J. Arima

January, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Formalization of Generalization-Based Analogy in General Logic Programs

Noboru IWAYAMA, Ken SATOH, Jun ARIMA

Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
email: iwayama@icot.or.jp

January 13, 1992

Abstract

Some research [3, 1, 2] has revealed logical aspects of analogy based generalization. Given similarity S and projected property P, inductive generalization gives a rule $\forall x.\ S(x) \supset P(x)$ from a fact $\exists x.\ S(x) \wedge P(x)$. In this paper, we show that a principle for analogy based on generalization is described and implemented in logic programs very clearly and simply. By the description, we give a declarative semantics of analogy based generalization. We also consider the implementation as being an analogical system by itself to support research on the analogy.

We need integrity constraints to describe the intrinsic non-deductiveness of the analogy. We consider the conclusion of analogical inference as one of minimal models which are not generally unique for a program. Therefore, stable model semantics is suitable for our formulation of the analogy. So, from the aspect of logic programming, this work gives us an important application of integrity constraints and a fine instance of utilization of stable model semantics.

We show the correspondence between an analogical model for a Horn logic program with integrity constraints and a stable model for an extended program of the original program. We also show the computation of the analogy by non-deterministic procedure of stable models.

Topic: Knowledge Representation

Category: Analogy, Logic Programming, Stable Model Semantics, Generalization, Integrity Constraint

1 Introduction

In this paper, we consider the analogy in the following sense: when two objects possess some common property (S: similarity), we would like to infer that a property (P: projected property) possessed by one object (B: base) is also possessed by another object (T: target).

Research on analogy may be divided into two types as stated by Peirce [12]. One is research to capture the analogy as a replacement of the base object with the target object (most research [18, 6, 9, 7, 10, 16] on the analogy are of this type). The other is research to capture analogy as a generalization from one instance [3]. Replacement is essential for the former type of the analogy. The replacement is to replace B satisfying S with T also satisfying S in order to obtain $P(T)$ "directly" from $P(B)$. In the latter type of analogy, inductive generalization is essential. that is to infer a general knowledge $\forall x.\ S(x) \supset P(x)$ (called the ***analogical grounded rule***) from a fact $S(B) \wedge P(B)$ ($\exists x.\ S(x) \wedge P(x)$). $P(B)$ is inferred from the analogical grounded rule and $S(T)$. We discuss the analogy from the latter point of view, namely, the analogy based on generalization.

So far advanced studies[3, 2] have been done in order to formalize analogy based on generalization from a logical point of view, it is important to crystallize fundamental units of analogy based on generalization. Since the declarative meanings of analogy based on generalization can strictly be expressed in logic programming, we believe that the description of the analogy based on generalization in logic programming is useful and promising. In this paper, we show that a principle for the analogy, provided by research on the analogy based on generalization, is described, and implemented in logic programming very clearly and simply. This implementation may also be considered as being an analogical system by itself to support research on the analogy as a logical inference.

Much work [10, 16, 9, 7] has dealt with analogy in logic programming. [10, 16] discussed analogy as a method to generate rules in the context of inductive logic programming. [9, 7] gave us an understanding of the analogy itself. Unlike those, this paper gives an implementation supported by a declarative semantics as a general tool for research on analogy based on generalization. Moreover, all of them have considered analogy as a replacement, while this paper deals with analogy based on generalization.

Now, we need further consideration to analogy in logic programming. Firstly, we discuss a relation between the non-deductiveness of analogy and logic programming. Previous implements of analogy in logic programming has not given enough attention to the non-deductiveness of analogy, though analogy is non-deductive intrinsically. The reason seems to be that previous systems have mostly implemented in a subset of the first order theory, logic programming (ex. Prolog), in practice[7, 9, 11], and that knowledge representations in these systems have also been restricted to the syntax of logic programming (definite clause[1]).

Let us consider non-deductive inference systems from a general point of view. We can consider the consequence of non-deductive inference as being a hypothesis, since the consequence is not necessarily deduced from given facts. In the process of scientific theory formation, the hypothesis should be verified. The verification must include the confirma-

[1] For *locally stratified* programs, including the class of definite clause, a clear semantics is given based on the ordinary Prolog interpreter with SLDNF strategy[13]. Unfortunately even locally stratified programs cannot help escaping the following argument.

tion that there is no counter example to hypothesis (the hypothesis is consistent to facts). Because scientists may revise or discard hypotheses to which there is a counter example, the refused hypotheses play a great part in acquiring the "correct" hypothesis. The use of counter example is very common to research in fields of inductive inference and learning. The point is that there is no way to prevent over-generalization except for counter examples, especially negative examples. The theory formation process that does not take into account negative examples becomes worthless, since the excessive generalization might explain any facts. However, we cannot represent negative examples in definite clause[2]. Therefore, research dealing with knowledge only in definite clause leave the verification process out of arguments from the logical standpoint. It follows from what has been said that the knowledge representation for analogy should be extended to logic programming which can deal with negative examples.

Recently, in the field of logic programming, much valuable research has enabled the extension of logic programming with clear logical semantics. One such extension of logic programming, *logic programs with integrity constraints*[14, 4], enables us to represent negative examples. So, in this paper, we discuss the analogy in logic programs with integrity constraints.

From the aspect of logic programming, this work gives us not only an important application of logic programs with integrity constraints but also of a fine instance of utilization of stable model semantics. We will consider the role of stable model semantics in the analogy in a little more detail. In this paper, to compute the analogy we will use general logic programs with integrity constraints that are not locally stratified programs. There are two semantics which deal with larger classes of logic programs beyond stratified programs: well-founded semantics [17] and stable model semantics [5]. We will consider the conclusion of analogical inference as one of minimal models which are not generally unique for a program. Intuitively speaking, while, in well-founded semantics, the union of all minimal models is considered as the meaning of the program, each minimal model is considered as the meaning of the program in stable model semantics. Therefore, we think that stable model semantics is suitable for our formulation.

To sum up, the main point stated above is that we give a formalization of analogy based generalization in general logic programs with integrity constraints. In the following section, we describe a principle for the analogy, which will be defined as an elementary form of analogical generalization. In section 3, we will translate it into a definition of the analogy which is based on the Herbrand model of a logic program with integrity constraints. In section 3 we show the correspondence between the definition of the analogy (Herbrand model) and stable model of the extended program of the original program. In section 4, we show the computation of the analogy by the non-deterministic procedure of stable models, previously given in [15].

[2]There are loopholes in some cases. Research on generation of definite clause program is in the field of inductive inference, in which this argument plays one of the key roles. In such a research, the technique to represent negative examples is some "meta" way, that is out of logic. But this way cannot strictly give logical meanings to negative examples (the inconsistency to the hypothesis).

2 Elementary Form of Analogical Generalization

In this section, we define a principle for analogy based on generalization.

A schema (1) is often used to explain the analogy. However, the prerequisite of the schema lacks some conditions [3, 1].

$$\frac{\begin{array}{l} S(B) \wedge P(B) \\ S(T) \end{array}}{P(T)} \tag{1}$$

Since the investigation of the implicit prerequisite in this schema is the heart of the research into the analogy, many researchers have devoted their efforts to this investigation. Researchers disagree on what the appropriate prerequisite is. Firstly, we provide the following two necessary prerequisites on which researchers, thinking of the analogy as a generalization from one instance, would agree:

Condition of Base Existence: $A \vdash \exists x.\ S(x) \wedge P(x)$, (2)

Grounded Rule Consistency: $A \cup \{\forall x.\ S(x) \supset P(x)\}$ is consistent. (3)

Condition (2) means that there must exist a base case, since analogical inference is based on the base case. Condition (3) means that the result of the generalization from one instance, the analogical grounded rule, must be consistent with the original knowledge to guarantee the consistency of the inference by the analogy. We note that the grounded rule consistency is equivalent to the following condition:

$$A \cup \{\neg(\exists x.\ S(x) \wedge \neg P(x))\} \text{ is consistent.} \tag{3'}$$

This condition shows that the entire knowledge is consistent with the original knowledge, even though there is no target object such that a conflicting result ($\neg P(T)$) is inferred by the analogy.

We define an *elementary form of analogical generalization* as an inference of the analogical grounded rule, $\forall x.\ S(x) \supset P(x)$, under the above two conditions. Moreover we say the above two conditions, condition of base existence and grounded rule consistency, as the necessary conditions for the analogy. In the following we discuss the analogy on the basis of the elementary form of analogical generalization.

3 Analogy in Logic programming

3.1 General Logic Program with Integrity Constraints

Firstly we define general logic programs with integrity constraints and their models. Those definitions are based on [14]. We should notice that the purpose of integrity constraints is different from the definition in [14]. While integrity constraints described in [14] are utilized when updating database to verify the consistency of the database, in this paper integrity constraints are utilized to verify whether the analogy is applicable.

Definition 3.1 *Let A be an atom, $L_1, ..., L_m (m \geq 0)$ be a literals.* A general logic program *is a set which consists of the following rules of the form:*

$$A \leftarrow L_1, L_2, ..., L_m.$$

or integrity constraints *of the form:*

$$\leftarrow L_1, L_2, ..., L_m.$$

We call A the *head* of the rule and $L_1, ..., L_m$ the *body* of the rule. Let R be a rule. We denote the head of R as $head(R)$, the set of positive literals in the body of R as $pos(R)$ and the set of atoms obtained by removing negation symbols from negative literals in the body of R as $neg(R)$. We say that a program that has no negative literal in bodies of rules nor integrity constraints is a *positive program*. In this paper, positive programs are subjects for the analogy while general logic programs are used for the computation of the analogy.

As stated in section 1, integrity constraints are used to represent negative examples. For example, an negative example $\neg P(T)$ (object T does not satisfy property P) is expressed as an integrity constraint $\leftarrow P(T)$. A general form of integrity constraint $\leftarrow L_1, L_2, ..., L_m$ means that there is at least one literal which is not satisfied.

The semantics for general logic programs is based on stable model semantics [5].

Definition 3.2 *Let K be a general logic program and θ be an arbitrary substitution of elements in the Herbrand base for all variables in K. A stable model M for K is a Herbrand model satisfying the following conditions.*

1. *M is equal to the minimal Herbrand model for the positive program K^M where*
   $K^M = \{R'\theta | head(R'\theta) = head(R\theta)$ *and* $pos(R'\theta) = pos(R\theta)$
   and $neg(R') = \emptyset$, *for* $R \in K$ *and* $neg(R\theta) \cap M = \emptyset\}$

2. *For every integrity constraint $C \in K$ and every substitution θ, $pos(C\theta) \not\subseteq M$ nor $(neg(C\theta) \cap M) \neq \emptyset$.*
   We say that M does not violate C.

3.2 Analogical Model and Analogical Schema

In the following we give definitions of the analogy for positive programs on the basis of the elementary form of analogical generalization defined in the previous section.

Definition 3.3 *Let K be a positive program, S, P be predicate symbols in K, $HU(K)$ be the Herbrand base of K and $min(K)$ be the least Herbrand model for K. A predicate P is* projectable *based on similarity S by analogy under K if the following conditions are satisfied:*

1. *There exists $t \in HU(K)$ s.t. $min(K) \models S(t) \wedge P(t)$,*

2. *There exists a Herbrand model M for K s.t. for any $t \in HU(K)$, $M \models S(t) \supset P(t)$.*

K satisfies the necessary conditions for the analogy described above, since the conditions of definition 3.3 correspond to the necessary conditions. Since K is a positive program, atoms satisfied by $min(K)$ are also satisfied by all models. The condition of base existence is given by this fact and the first condition of definition 3. The second condition is equivalent to grounded rule consistency.

Definition 3.4 *Let S, P be predicate symbols in positive program K. P be projectable based on similarity S by analogy under K. A Herbrand model M for K is an* analogical model *for K on P with similarity S iff M is the least model for $K \cup \{P(x) \leftarrow S(x)\}$.*

Although this definition is a 'meta' definition with respect to the Herbrand model for K, we can define the analogy as a stable model for the following extended program of K.

Definition 3.5 *Let K be a positive program, S, P be predicate symbols in K, $appli, contra$ be predicate symbols not in K. The following set of rules is the* analogical generalization schema *on P with similarity S, expressed as $\mathcal{A}(S, P)$:*

$$P(x) \leftarrow \neg contra, appli, S(x).$$

$$appli \leftarrow S(x), P(x).$$

$$contra \leftarrow S(x), \neg P(x).$$

In the schema, *appli* is true if the condition of base existence is satisfied, while *contra* is true if grounded rule consistency is not satisfied. When *appli* is true and *contra* is false,. the first rule of the schema is equivalent to the analogical grounded rule. Therefore, we may say that this schema represents the elementary form of analogical generalization.

Definition 3.6 *Let K be a positive program, S, P be predicate symbols in K. An* analogical extended program *of K is a general logic program $K \cup \mathcal{A}(S, P)$. When S and P are identified from the context, we express $K \cup \mathcal{A}(S, P)$ as K_+.*

The following two theorems show the correspondence between analogical models and stable models for an analogical extended program (for proofs, see appendix).

Theorem 3.7 *Let K be a positive program. If there exists a stable model M for K_+ such that $appli \in M$ and $contra \notin M$, then P is projectable based on similarity S, and $M - \{appli\}$ is an analogical model for K on P with similarity S.*

Theorem 3.8 *Let K be a positive program. If P is projectable based on similarity S by the analogy under K, and M is an analogical model for K, then $M \cup \{appli\}$ is a stable model for K_+.*

It follows from these theorems that we can obtain an analogical model for K by computing a stable model M for K_+, because $M - \{appli\}$ is an analogical model for K if and only if $appli \in M$ and $contra \notin M$.

Since K_+ is not a locally stratified program, a stable model for K_+ corresponding to an analogical model for K is a minimal model that is not generally unique. As described in section 1, stable model semantics gives us a key to grasp the analogy based on generalization. The following example makes this clear.

Example 3.9 *Relation between Analogical model and Stable Model*

When there is an analogical model for K, there is probably a stable model for K_+ which does not correspond to the analogical model. Consider the following program K:

$S(B)$.
$P(B)$.
$S(T)$.

There are two stable models for K_+ of this program: $\{S(B), P(B), S(T), P(T), appli\}$ and $\{S(B), P(B), S(T), appli, contra\}$. The first model corresponds to the analogical model for K_+ because in the first model *appli* is included and *contra* is not included. On the other hand, the well-founded model for K_+ is $\{S(B), P(B), S(T), appli\}$. This model cannot represent the possibility that $P(T)$ is satisfied. We can see from this example that stable model semantics provide us with the ability to select the analogical model from minimal models.

4 Computing the Analogy

In this section, we provide a way of computing the analogy using a nondeterministic bottom-up procedure. To terminate the procedure, we deal with general logic programs satisfying the following conditions:

1. There is no function symbol.
   (There are constant symbols only.)

2. All rules are range-restricted.
   (All variables in the head of each rule must occur in the body.)

During the computation, the bottom-up procedure uses ground instances of rules in general logic programs, thanks to the above conditions. Now we will show a nondeterministic procedure computing stable models. The procedure is an expansion of procedure [15] which is a bottom-up procedure that computes stable models for propositional general logic programs with integrity constraints.

A procedure computing stable models
Let K be a general logic program and θ be an arbitrary substitution of elements in the Herbrand base for all variables in K.

$i := 0$,
$M_0, \widetilde{M}_0 := propagate(\emptyset, \emptyset)$.
If $M_0 \cap \widetilde{M}_0 \neq \emptyset$ then **fail**.

Step 1:
Select a rule R and θ in K such that $head(R\theta) \notin M_i$ and $pos(R\theta) \subseteq M_i$ and $(neg(R\theta) \cap M_i) = \emptyset$, then go to **Step 2**.
If such a rule is not found and there exists an integrity constraint C in K and θ s.t. M_i violates $C\theta$
then **fail** else **return** M_i.

Step 2:
$i := i + 1$,
$M_i, \widetilde{M}_i := propagate(M_{i-1} \cup \{head(R)\}, \widetilde{M}_{i-1} \cup neg(R))$

If $M_i \cap \widetilde{M}_i \neq \emptyset$ then **fail** else go to **Step 1**.

propagate$(M, \widetilde{M})$
begin
$k := 0$, $M_i^0 := M$, $\widetilde{M}_i^0 := \widetilde{M}$.
do
$k := k + 1$, $M_i^k := M_i^{k-1}$, $\widetilde{M}_i^k := \widetilde{M}_i^{k-1}$.

For every rule R in K

1. If there is a θ such that $head(R\theta) \notin M_i^{k-1}$ and $pos(R\theta) \subseteq M_i^{k-1}$ and $neg(R\theta) \subseteq \widetilde{M}_i^{k-1}$, then add $head(R\theta)$ to M_i^k.
2. If there is a θ such that $head(R\theta) \in \widetilde{M}_i^{k-1}$ and there exists $P \in pos(R\theta)$ s.t. $P \notin M_i^{k-1}$ and $(pos(R\theta) - \{P\}) \subseteq M_i^{k-1}$, and $neg(R\theta) \subseteq \widetilde{M}_i^{k-1}$ then add P to $\widetilde{M}_i^k$.
3. If there is a θ such that $head(R\theta) \in \widetilde{M}_i^{k-1}$ and $pos(R\theta) \subseteq M_i^{k-1}$ and $neg(R\theta) \subseteq \widetilde{M}_i^{k-1}$, then **fail**.

For every integrity constraint C in K.

4. If there is a θ such that there exists $P \in C\theta$ s.t. $P \notin M_i^{k-1}$ and $(C\theta - \{P\}) \subseteq M_i^{k-1}$, then add P to $\widetilde{M}_i^k$.
5. If there is a θ such that $C\theta \subseteq M_i^{k-1}$, then **fail**.

until $M_i^k = M_i^{k-1}$ **and** $\widetilde{M}_i^k = \widetilde{M}_i^{k-1}$.
return $M_i^k, \widetilde{M}_i^k$
end

In the procedure, *select* in Step 1 expresses nondeterminism and *fail* expresses going back to the recent choice point. For the range-restricted programs without function symbols as stated above, the output of the procedure is a stable model (sound) and the procedure outputs all stable models by exhaustive search (complete). See [15] for a proof of soundness and completeness of the procedure.

Example 4.1 *Computation of the Analogy*

Consider the following program K :

$S(B)$. (1)
$P(B)$. (2)
$S(T)$. (3)

We will add the following analogical generalization schema $\mathcal{A}(S, P)$ to the above program.

$P(x) \leftarrow \neg contra, appli, S(x)$. (4)
$appli \leftarrow S(x), P(x)$. (5)
$contra \leftarrow S(x), \neg P(x)$. (6)

We show how models for $K_+ (= K \cup \mathcal{A}(S, P))$ are constructed by each selection of the rules in the above procedure.

Selection 1.

0. $M_0 = \{S(B), P(B), S(T), appli\}$, $\widetilde{M}_0 = \emptyset$

1. Select (6) with $\theta = \{x = T\}$.
   $M_1 = \{S(B), P(B), S(T), appli, contra\}$, $\widetilde{M}_1 = \{P(T)\}$

2. Since there is no selected rule, M_1 is returned as a stable model. However, M_1 does not correspond to an analogical model because the model includes *contra*.

Selection 2.

0. $M_0 = \{S(B), P(B), S(T), appli\}$, $\widetilde{M}_0 = \emptyset$

1. Select (4) with $\theta = \{x = T\}$.
   $M_1 = \{S(B), P(B), S(T), appli, P(T)\}$, $\widetilde{M}_1 = \{contra\}$

2. Since there is no selected rule, M_1 is returned as a stable model. $M_1 - \{appli\}$ is an analogical model because *appli* is included but *contra* is not included in M_1. It follows from the analogy that $P(T)$ is satisfied in M_1.

Example 4.2 *Effect of the Integrity Constraint*

Consider the following program K which includes integrity constraint (4):

$S(B)$. (1)
$P(B)$. (2)
$S(T)$. (3)
$\leftarrow R(T)$. (4)
$R(x) \leftarrow P(x)$. (5)

We will add the following analogical generalization schema $\mathcal{A}(S, P)$ to the above program.

$P(x) \leftarrow \neg contra, appli, S(x)$. (6)
$appli \leftarrow S(x), P(x)$. (7)
$contra \leftarrow S(x), \neg P(x)$. (8)

We show how models for K_+ are constructed by each selection of the rules in the above procedure.

Selection 1.

0. $M_0 = \{S(B), P(B), S(T), appli, contra\}$, $\widetilde{M}_0 = \{R(T), P(T)\}$

1. Since there is no selected rule, M_0 is returned as a stable model. M_0 does not correspond to the analogical model for K because M_0 includes *contra*.

Since there is no selection of rules, there is no analogical model for K_+. The reason for this is that the grounded rule consistency is not satisfied by (4).

Example 4.3 *Relaxation of the Restriction for S and P*

In the definitions for the analogical model and the analogical generalization schema, predicates' arity is one. This restriction is not critical for our analogy. In definitions for the analogical model and the analogical generalization schema, predicate S may be replaced by a conjunction of predicates and predicate P may be a predicate with any arity. The following serves as an example. Consider the following program K:

$ApartFrom(Ball, Block)$. (1)
$ApartFrom(Planet, Sun)$. (2)
$ApartFrom(Electron, Nucleus)$. (3)
$Attracts(x, y) \leftarrow AstroHeavy(x), Object(y)$. (4)
$Attracts(x, y) \leftarrow PosElect(x), NegElect(y)$. (5)
$Attracts(x, y) \leftarrow NegElect(x), PosElect(y)$. (6)
$AstroHeavy(Sun)$. (7)
$Object(Planet)$. (8)
$PosElect(Nucleus)$. (9)
$NegElect(Electron)$. (10)
$Revolves(Planet, Sun)$. (11)
$\leftarrow Revolves(Ball, Block)$. (12)

We will show whether $Revolves(Electron, Nucleus)$ ("electron revolves round nucleus"), which is a non-deductive consequence, can be inferred by the analogy for different candidates of similarity. To do so, we consider $Revolves(x, y)$ as projected property $P(x, y)$ and $\langle Electron, Nucleus \rangle$ as a target object T.

Case 1. A candidate for similarity $S(x, y)$ is $NegElect(x) \wedge PosElect(y)$: We will add the following analogical generalization schema to K.

$Revolves(x, y) \leftarrow \neg contra, appli, NegElect(x), PosElect(y)$. (13)
$appli \leftarrow NegElect(x), PosElect(y), Revolves(x, y)$. (14)
$contra \leftarrow NegElect(x), PosElect(y), \neg Revolves(x, y)$. (15)

We show the computation by each selection of the rules in the procedure. We use M and $\widetilde{M}$ for convenience:
$M = \{ApartFrom(Ball, Block), ApartFrom(Planet, Sun), ApartFrom(Electron, Nucleus),$
$AstroHeavy(Sun), Object(Planet), PosElect(Nucleus), NegElect(Electron), Revolves(Planet, Sun),$
$Attracts(Sun, Planet), Attracts(Nucleus, Electron), Attracts(Electron, Nucleus)\}$,
$\widetilde{M} = \{Revolves(Ball, Block)\}$.

Selection 1.

0. $M_0 = M$, $\widetilde{M}_0 = \widetilde{M}$
1. Select (15) with $\theta = \{x = Electron, y = Nucleus\}$.
   $M_1 = M_0 \cup \{contra\}$, $\widetilde{M}_1 = \widetilde{M}_0 \cup \{Revolves(Electron, Nucleus)\}$.
   Since there is no selected rule, M_1 is returned as a stable model. M_1 does not correspond to the analogical model for K because M_1 includes *contra*.

Since there is no selection of rules, there is no analogical model for K_+. The reason for this is that the condition of base existence is not satisfied by the fact that there is no object which satisfies $(NegElect(x) \wedge PosElect(y)) \wedge Revolves(x, y)$ ("a negatively-charged object revolves round a positively-charged object").

Case 2. A candidate for similarity $S(x, y)$ is $ApartFrom(x, y)$: We will add the following analogical generalization schema to K.

$Revolves(x, y) \leftarrow \neg contra, appli, ApartFrom(x, y)$. (16)
$appli \leftarrow ApartFrom(x, y), Revolves(x, y)$. (17)
$contra \leftarrow ApartFrom(x, y), \neg Revolves(x, y)$. (18)

We show the computation by each selection of the rules in the procedure.

Selection 1.

0. $M_0 = M \cup \{appli, contra\}$, $\widetilde{M}_0 = \widetilde{M}$

1. Since there is no selected rule, M_0 is returned as a stable model. M_0 does not correspond to the analogical model for K because M_0 includes *contra*.

Since there is no selection of rules, there is no analogical model for K_+. The reason for this is that the grounded rule consistency is not satisfied by the fact that there is a counter example against the analogical grounded rule. $ApartFrom(Ball, Block) \wedge \neg Revolves(Ball, Block)$ ("though *Ball* is apart from *Block*, *Ball* does not revolve round *Block*").

Case 3. A candidate for similarity $S(x, y)$ is $ApartFrom(x, y) \wedge Attracts(y, x)$: We will add the following analogical generalization schema to K.

$$Revolves(x, y) \leftarrow \neg contra, appli, ApartFrom(x, y), Attracts(y, x). \quad (19)$$
$$appli \leftarrow ApartFrom(x, y), Attracts(y, x), Revolves(x, y). \quad (20)$$
$$contra \leftarrow ApartFrom(x, y), Attracts(y, x), \neg Revolves(x, y). \quad (21)$$

We show the computation by each selection of the rules in the procedure.

Selection 1.

0. $M_0 = M \cup \{appli\}$, $\widetilde{M}_0 = \widetilde{M}$

1. Select (21) with $\theta = \{x = Electron, y = Nucleus\}$. $M_1 = M_0 \cup \{contra\}$, $\widetilde{M}_1 = \widetilde{M}_0 \cup \{Revolves(Electron, Nucleus)\}$.
   Since there is no selected rule, M_1 is returned as a stable model. M_1 does not correspond to the analogical model for K because M_1 includes *contra*.

Selection 2.

0. $M_0 = M \cup \{appli\}$, $\widetilde{M}_0 = \widetilde{M}$

1. Select (19) with $\theta = \{x = Electron, y = Nucleus\}$.
   $M_1 = M_0 \cup \{Revolves(Electron, Nucleus)\}$, $\widetilde{M}_1 = \widetilde{M}_0 \cup \{contra\}$.
   Since there is no selected rule, M_1 is returned as a stable model. $M_1 - \{appli\}$ is an analogical model because *appli* is included but *contra* is not included in M_1. It follows from the analogy that $Revolves(Electron, Nucleus)$ is satisfied in M_1. This computation process corresponds to the following inference of $Revolves(Electron, Nucleus)$: after the analogical grounded rule,

$$\forall x, y.\ Attracts(y, x) \wedge ApartFrom(x, y) \supset Revolves(x, y)$$

   ("if y attracts x and x is apart from y, then x revolves round y"), is obtained on the basis of the elementary form of analogical generalization,

$$Attracts(Nucleus, Electron) \wedge ApartFrom(Electron, Nucleus)$$

   (" nucleus attracts electron and electron is apart from nucleus") is inferred.

5 Related Work

Orihara et al. [11] proposed the analogy on the basis of stable model (generalized stable model precisely). They discussed the analogy for locally stratified programs without integrity constraints. Since [11] added extra rules to the original program to compute generalized stable models representing analogical models, their formulation is similar to our approach. However, there are two main differences between their approach and ours, as follows:

1. Negative Information

   [11] used locally stratified programs for knowledge representation. As we described in section 1, negative information is not described in locally stratified programs. Therefore [11] used heuristics or justification by the user to select a plausible analogical model. Because our formulation uses integrity constraints, we can control analogical inference only in a logical framework.

2. Result of the Analogy

   [11] does not distinguish similarity from projected property to infer as many facts as possible. For example, the analogical model intended in [11] for a model $\{S(B), P(B), S(T_1), P(T_2)\}$ is a model $\{S(B), P(B), S(T_1), P(T_1), S(T_2), P(T_2)\}$. Since our analogical model is a model $\{S(B), P(B), S(T_1), P(T_1), P(T_2)\}$, a fact $S(T_2)$ is not obtained by the analogy. The reason is that [11] added the following two rules to the original program: a rule that corresponds to the analogical grounded rule $\forall x.\ S(x) \supset P(x)$, and a rule that corresponds to a rule $\forall x.\ P(x) \supset S(x)$, the reverse of analogical grounded rule.

6 Concluding Remarks

In the future, we will consider the following two points. The first is concerned with the nature of analogical inference. The second takes the standpoint of logic programming.

The solution of central problems have been important in the study of analogy: which object should be selected as a base with respect to a target, which property is important in the analogy among properties shared by two objects, and which property is to be projected with respect to a certain similarity. These problems are all concerned with similarity and projected property. However, in our formulation, similarity and projected property should be specified before the computation. It follows that we cannot deal with how to specify similarity and projected property in logic. We have just found a clue to this difficulty. That is a modification for the analogical generalization schema:

$$U(x,p) \leftarrow \neg contra(s,p), appli(s,p), C(x,s).$$

$$appli(s,p) \leftarrow C(x,s), U(x,p).$$

$$contra(s,p) \leftarrow C(x,s), \neg U(x,p).$$

In this modified schema, $C(x,s)$ means that the value of attribute C in object x is s. We introduce this notation to avoid the use of second order languages. Because, using the

above schema, we would specify similarity and projected property in logic programming, we will consider more about the schema.

From the standpoint of logic programming, there are two points to be considered. The first is the expansion of the class of the original programs for the analogy. In this paper, the class of the original programs is Horn logic. We would like to deal with general logic programs as the class of the original programs. The second is the utilization of a top-down procedure to answer queries efficiently. Our bottom-up procedure presents the possibility of computing irrelevant models to the query. We think that a top-down procedure in [8] may be useful.

Appendix

Proof of Theorem 3.7: A stable model M for a program K_+ is a minimal model for a program K_+^M, that is $K \cup \{P(t) \leftarrow appli, S(t) | t \in HU(K)\} \cup \{appli \leftarrow S(u), P(u) | u \in HU(K)\} \cup \{contra \leftarrow S(v) | v \in HU(K) \text{ and } P(v) \notin M\}$, and M is consistent with the integrity constraint in K. So $M - \{appli\}$ is a model for K.
Suppose P is not projectable.

1. Suppose that there does not exist $t \in HU(K)$ s.t. $min(K) \models S(t) \wedge P(t)$.
   Since $appli \in M$, $appli$ should be inferred from K_+^M. $S(u) \wedge P(u)$ is not inferred from K or other rules. This is a contradiction.

2. Suppose that for some $t \in HU(K)$, $M - \{appli\} \not\models S(t) \supset P(t)$.
   Since there exists $t \in HU(K)$ s.t. $M \models S(t) \wedge \neg P(t)$, $contra$ is inferred. However, this contradicts the fact that $contra \notin M$.

Since $appli$ is a predicate symbol not in K, and M is the least model for K_+^M, $M - \{appli\}$ is the least model for $K \cup \{P(x) \leftarrow S(x)\}$. □

Proof of Theorem 3.8:
Let an analogical model for K be M.
We show that $m = M \cup \{appli\}$ is a stable model for K_+. In other words, we show that $m = min(K_+^m)$.

Since $contra$ is a predicate symbol not in K, $contra \notin m$ and $K_+^m = K \cup \{P(t) \leftarrow appli, S(t) | t \in HU(K)\} \cup \{appli \leftarrow S(u), P(u) | u \in HU(K)\} \cup \{contra \leftarrow S(v) | v \in HU(K) \text{ and } P(v) \notin M\}$.

1. Since $M \subseteq m$, m satisfies K.

2. Since for any $t \in HU(K)$, $M \models P(t) \leftarrow S(t)$, $m (= M \cup \{appli\})$ satisfies $\{P(t) \leftarrow appli, S(t) | t \in HU(K)\}$.

3. Since $appli \in m$, m satisfies $\{appli \leftarrow S(u), P(u) | u \in HU(K)\}$.

4. Since for any $t \in HU(K)$, $M \models P(t) \leftarrow S(t)$, $S(v) \notin M$ if $P(v) \notin M$. Therefore m satisfies $\{contra \leftarrow S(v) | v \in HU(K) \text{ and } P(v) \notin M\}$.

Therefore, m is a model for K_+^m.

Since for any $t \in HU(K)$, $min(K) \models S(t) \wedge P(t)$ *appli* $\in min(K_+^m)$. M is the least model for $K \cup \{P(x) \leftarrow S(x)\}$. From these facts, we obtain that $m \supseteq min(K_+^m)$. So we conclude $m = min(K_+^m)$ from the minimality of $min(K_+^m)$.

M does not violate integrity constraints in K, since M is an analogical model for K. Therefore, m is a stable model for K_+. □

References

[1] Arima, J., A Logical Analysis of Relevance in Analogy, in *Proc. of ALT'91*, Japanese Society for Artificial Intelligence, pp. 255 - 265 (1991).

[2] Arima, J., Logical Structure of Analogy, to appear in *Int. Conference on FGCS'92* (1992).

[3] Davies, T., Russell, S.J., A Logical Approach to Reasoning by Analogy, *Proc. of IJCAI-87* pp. 264 - 270 (1987).

[4] Eshghi, K., Kowalski, R. A., Abduction Compared with Negation by Failure, *Proc. of ICLP'89*, pp. 234 - 254 (1989).

[5] Gelfond, M., Lifschitz, V., The Stable Model Semantics for Logic Programming, *Proc. of LP'88*, pp. 1070 - 1080 (1988).

[6] Gentner, D., Structure-mapping: Theoretical Framework for Analogy, *Cognitive Science*, Vol.7, No.2 pp. 155 - 170 (1983)

[7] Haraguchi, M., Arikawa, S., Partial identity between least Herbrand models of logic programs, *Proc. of the Int. Workshop on Analogical and Inductive Inference '86*, Springer LNCS 265, pp. 61 - 87 (1987).

[8] Kakas, A. C., Mancarella, P., On the Relation between Truth Maintenance and Abduction, *Proc. of PRICAI'90*, pp. 438 - 443 (1990).

[9] Kedar-Cabelli, S., Purpose-Directed Analogy, *Proc. of the 7th Annual Conference of the Cognitive Science Society*, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 150 - 159 (1985).

[10] Kodratoff, Y., Using Abductive Recovery of Failed Proofs for Problem Solving by Analogy, *Proc. of the 7th Int. Conference of Machine Learning* , pp. 295 - 303 (1990).

[11] Orihara, R., Analogical Reasoning as a Form of Hypothetical Reasoning and Justification-based Knowledge Acquisition, in *Proc. of ALT'91*, Japanese Society for Artificial Intelligence, pp. 243 - 254 (1991).

[12] Peirce, C.S., *Elements of Logic*, in:Hartshorne, C., and Weiss, P. (eds.), *Collected Papers of Charles Sanders Peirce*, Volume 2 (Harvard University Press, Cambridge, MA, 1932).

[13] Przymusinski,T.C., On the Declarative Semantics of Deductive Databases and Logic Programs, *Foundations of Deductive Databases and Logic Programs* (J. Minker, ed.), Morgan Kaufman, Chapter 5, pp. 193 – 216 (1988).

[14] Sadri, F., Kowalski, R., A Theorem-Proving Approach to Database Integrity, *Foundations of Deductive Databases and Logic Programs* (J. Minker, ed.), Morgan Kaufman, Chapter 9, pp. 313 – 362 (1988).

[15] Satoh, K., Iwayama, N., Computing Abduction by Using the TMS, *Proc. of ICLP'91*, pp. 505 – 518 (1991).

[16] Tausend, B., Bell, S., Analogical Reasoning for Logic Programming, *Proc. of Int. Workshop on Inductive Logic Programming*, pp. 159 – 165 (1991).

[17] Van Gelder, A., Ross, K.A., Schlipf, J.S., The Well-Founded Semantics for General Logic Programs, *Journal of ACM*, Vol.38, No,3, pp. 620 – 650 (1991).

[18] Winston,P.H.: Learning Principles from Precedents and Exercises, *Artificial Intelligence*, Vol. 19, No. 3 (1982).