A Note on Rule-Finding Abduction

Hirata, Koichi
Research Institute of Fundamental Information Science, Kyushu University

http://hdl.handle.net/2324/3077
A Note on Rule-Finding Abduction

Kouichi Hirata
Research Institute of Fundamental Information Science,
Kyushu University 33, Fukuoka 812, Japan
e-mail: hirata@rifs.kyushu-u.ac.jp

Abstract

The rule-finding abduction is an abduction which begins with the observation of a surprising fact, finds a rule in the set of programs, and proposes a hypothesis. This paper investigates such rule-finding abduction for logic programming from two viewpoints: termination and analogical reasoning. In order to discuss the termination of rule-finding abduction, we introduce two concepts of loop-pair and loop-elimination. We show that the loop-pair is a syntactical condition to determine whether the process of rule-finding abduction is infinite. Also we show that, by using loop-elimination, we can delete all the infinite processes of rule-finding abduction. On the other hand, we discuss the relationship between analogical reasoning and rule-finding abduction. Then, we formulate rule-finding abduction with analogy, and define a deducible hypothesis. We show that a deducible hypothesis is correct in the sense of analogical reasoning. Also we show that, if a target program is empty, then a deducible hypothesis is polynomial time computable on the length of a surprising fact and the size of a proof tree. Furthermore, we realize rule-finding abduction with analogy by a Prolog program.

1 Introduction

Abduction [Pei65, Yon82] is the first stage of scientific inquiry, and also the methodology to discover new concepts. It begins with an observation of a surprising fact, and proposes a hypothesis to explain why the fact arises. An inference schema by abduction was described by the following three steps [Pei65, Yon82].

1. A surprising fact $C$ is observed.

2. If $A$ were true, then $C$ would be a matter of course.

3. Hence, there is reason to suspect that $A$ is true.

In general, the above inference schema is depicted by a syllogism:

$$\begin{array}{c}
C \\ A \rightarrow C \\ \hline
A
\end{array}$$

In computer science, various researchers have dealt with and formulated abduction. We can classify abduction for logic programming into five types [Hir93]. The rule-finding abduction is an abduction which finds a rule in the set of logic programs and proposes a hypothesis to explain a surprising fact. Here, the set of logic programs is given in advance. In this paper,
we investigate rule-finding abduction from two viewpoints: the termination and analogical aspects.

Since abduction is the first stage of scientific inquiry, we hope that the processes of abduction are finite. Furthermore, since the set of logic programs is given in advance, we deal with the combination of the programs in order to discuss the termination of rule-finding abduction. Hence, we pay our attention to the infinite process which is caused by the combination of programs. In this paper, we introduce two concepts, loop-pair and loop-elimination. For the combination of programs, a loop-pair determines whether the process of rule-finding abduction is infinite. On the other hand, the loop-elimination is a transformation of a program. By using loop-elimination, we can delete all the infinite processes of rule-finding abduction.

Analogy [Pol54a, Pol54b, Pol57], which is also a kind of methodology to solve a problem, is an inference based on similarity between objects. Haraguchi and Arikawa [Har85, HaA86, HiA94b] have formulated such analogy mathematically, and have pointed out that it is an important tool for machine learning. It acquires unknown analogical facts in domains by computing an analogy which gives a similarity between the domains.

In order to discuss the relationship between rule-finding abduction and analogical reasoning, we formulate rule-finding abduction with analogy, which is an extension of rule-finding abduction. Also we define a deducible hypothesis which is correct as the sense of analogical reasoning. The problem of abduction with analogy is how to detect an analogy \( \varphi \). By using partial isomorphic generalization [HiA94b], we give an algorithm for abduction with analogy, and realize it as a Prolog program.

This paper is organized as follows: In Section 2, we prepare the notions of head-reducing programs to characterize the termination of abduction. In Section 3, we explain what rule-finding abduction is. In Section 4, 5, and 6, we discuss the termination of rule-finding abduction. In Section 4, we present the concept of abducible predicate, which has been introduced in the field of abductive logic programming. In Section 5, we introduce the concept of loop-pair. We show that if the loop-pair appears in a derivation, then the derivation becomes infinite. In Section 6, we introduce the concept of loop-elimination, which is a kind of transformations of a program. We also show that, for given two programs, if we transform one program by loop-elimination, then all the derivations of union of the transformed program and the rest are finite. In other words, by loop-elimination, we can choose the programs whose proof trees have no infinite branches. In Section 7, we prepare the concepts of analogical reasoning and partially isomorphic generalization. In Section 8, we formulate rule-finding abduction with analogy and introduce a deducible hypothesis, which is an extension of hypotheses of rule-finding abduction. We show that, if a target program is empty, then a deducible hypothesis is polynomial time computable on the length of a surprising fact and the size of proof tree. Also we design an algorithm for it concretely, and realize it by a Prolog program.

2 Preliminary

In this section, we introduce the concept of head-reducing program [Hir93] to characterize the termination of abduction.

First, we introduce the following definitions.

**Definition 1** Let \( P \) be a definite program and \( p \) be a predicate symbol. Then, a recursive definition of \( p \) for \( P \), denoted by \( \text{rec}(P, p) \), is a definition clause of \( p \) defined by the following procedure:

1. Select a clause in \( P \) whose head has the predicate \( p \), and let it be \( \text{rec}(P, p) \).
2. For $\text{rec}(P, p) = A \leftarrow B_1, \ldots, B_l, \ldots, B_n$, if there exists a clause $E \leftarrow F_1, \ldots, F_m$ such that $B_i \theta = E \theta$ for a substitution $\theta$, and $\text{pred}(B_i)(= \text{pred}(E)) \neq p$, then eliminate the clause $E \leftarrow F_1, \ldots, F_m$ from $P$, and put

\[ \text{rec}(P, p) = (A \leftarrow B_1, \ldots, B_{l-1}, F_1, \ldots, F_m, B_{l+1}, \ldots, B_n) \theta. \]

3. Repeat 2 until it cannot be applied.

A recursive program of $p$ for $P$, denoted by $R(P, p)$, is a program consisting of a recursive definition $\text{rec}(P, p)$ and the applied clauses in constructing $\text{rec}(P, p)$.

For a definite program $P$ and a predicate symbol $p$, $\text{rec}(P, p)$ and $R(P, p)$ are not unique in general.

A clause $p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$ is said to be $p$-reducing with respect to the $i$-th argument if $|t_i \theta| > |s_j^i \theta|$ for any substitution $\theta$ and for any index $l$ such that $\text{pred}(B_l) = p$, where $s_j^i$ is the $i$-th argument’s term of $B_l$. The $s_j^i$ is called $p$-reducing term. A $p$-reducing clause with respect to some argument is called a $p$-reducing clause simply. These definitions are the extensions of reducing and weakly reducing programs by Yamamoto [Yam92].

Whether or not the process of rule-selecting abduction for a definite program terminates is characterized as the following concept of head-reducing.

**Definition 2** Let $\text{rec}(P, p)$ be a recursive definition $p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_m$ of $p$ for $P$. Then, a recursive program $R(P, p)$ is called head-reducing if it satisfies the following conditions:

1. If there exists an index $k$ such that $B_k = p(s_1^k, \ldots, s_n^k)$, then
   a. there exists an index $j$ such that $|t_j \theta| > |s_j^k \theta|$ for any such $k$ and for any substitution $\theta$, and
   b. any atom $B_l$ such that $B_l = q_l(w_1^l, \ldots, w_n^l)$ ($p \neq q_l$) satisfies one of the following conditions:
      i. there exists the $i$-th argument’s term $w_i^l$ in $B_l$ which is constructed by the variables appearing in $t_j$, and the definition clause of $q_l$ is $q_l$-reducing with respect to the $i$-th argument, or
      ii. the definition clause of $q_l$ is not included in $R(P, p)$.

2. Otherwise, any $B_l = q_l(w_1^l, \ldots, w_n^l)$ satisfies one of the following conditions:
   a. there exists the $i$-th argument’s term $w_i^l$ in $B_l$ which is constructed by the variables appearing in all arguments’ terms $t_1, \ldots, t_n$ in $p(t_1, \ldots, t_n)$, and the definition clause of $q_l$ is $q_l$-reducing with respect to the $i$-th argument, or
   b. the definition clause of $q_l$ is not included in $R(P, p)$.

Furthermore, $P$ is head-reducing with respect to the predicate $p$ if any recursive program $R(P, p)$ of $p$ for $P$ is head-reducing.

**Theorem 1** Let $P$ be a definite program and $p$ be a predicate symbol. If $P$ is head-reducing with respect to $p$, then all the SLD-derivations of $P \cup \{ \leftarrow p(s_1, \ldots, s_n) \}$ are finite.
applied programs hypotheses
nothing \( H_0 = \{ \text{find}(i, \text{fossil\_shell}, \text{mountain}) \} \)
\( P_1 \) \( H_1 = \{ \text{sea}(\text{mountain}) \} \)
\( P_2 \) \( H_2 = \{ \text{be}(\text{mountain}, \text{sea}) \} \)
\( P_3 \) \( H_3 = \{ \text{used\_to\_be}(\text{mountain}, \text{sea}) \} \)
\( P_4 \) \( H_4 = \{ \text{has\_not\_leg}(\text{fossil\_shell}) \} \)
\( P_5 \) \( H_5 = \{ \text{has\_not\_wing}(\text{fossil\_shell}) \} \)
\( P_6 \) \( H_6 = \{ \text{slow\_move}(\text{fossil\_shell}, \text{sea}, \text{mountain}) \} \)
\( P_7 \) \( H_7 = \{ \text{move}(\text{fossil\_shell}, \text{sea}, \text{mountain}) \} \)

Figure 1: Applied programs and hypotheses \( H_i \) of \( \alpha \) for \( \{ P_1, P_2, P_3 \} \)

3 Rule-Finding Abduction for Logic Programming

Consider the following fossil-shell example. Let \( P_i \) \((1 \leq i \leq 3)\) be the following sets of clauses:

\[
P_1 = \left\{ \begin{array}{l}
\text{find}(X, \text{fossil\_shell}, Y) \leftarrow \text{sea}(Y) \\
\end{array} \right\},
\]

\[
P_2 = \left\{ \begin{array}{l}
\text{find}(X, \text{fossil\_shell}, Y) \leftarrow \text{used\_to\_be}(Y, \text{sea}) \\
\text{used\_to\_be}(X, Y) \leftarrow \text{be}(X, Y) \\
\end{array} \right\},
\]

\[
P_3 = \left\{ \begin{array}{l}
\text{find}(X, \text{fossil\_shell}, Y) \leftarrow \text{move}(\text{fossil\_shell}, \text{sea}, Y) \\
\text{move}(\text{fossil\_shell}, X, Y) \leftarrow \text{slow\_move}(\text{fossil\_shell}, X, Y) \\
\text{slow\_move}(X, Y, Z) \leftarrow \text{has\_not\_leg}(X) \\
\text{slow\_move}(X, Y, Z) \leftarrow \text{has\_not\_wing}(X) \\
\end{array} \right\}.
\]

Let \( \alpha \) be a surprising fact \( \text{find}(i, \text{fossil\_shell}, \text{mountain}) \). In rule-finding abduction, by selecting a program \( P_i \) from the set \( \{ P_1, P_2, P_3 \} \) of programs, we find a rule and propose a hypothesis which explains a surprising fact \( \alpha \). We call such a selected program an applied program of \( \alpha \) for \( \{ P_1, P_2, P_3 \} \). Then, Figure 1 illustrates the applied programs and hypotheses \( H_i \) of \( \alpha \) for \( \{ P_1, P_2, P_3 \} \). For the applied program \( P_1, P_1 \cup H_1 \vdash \alpha \). For the applied program \( P_2, P_2 \cup H_2 \vdash \alpha \) and \( P_3 \cup H_3 \vdash \alpha \). For the applied program \( P_3, P_3 \cup H_i \vdash \alpha \) \((4 \leq i \leq 7)\). The hypothesis \( H_6 \) is a trivial hypothesis, that is, \( H_0 \vdash \alpha \).

Furthermore, we can give the example of programs with function symbols and recursion. Let \( P_i \) \((4 \leq i \leq 6)\) be the following sets of clauses:

\[
P_4 = \left\{ \begin{array}{l}
p(f(f(X))) \leftarrow p(X), q(f(X)) \\
\end{array} \right\},
\]

\[
P_5 = \left\{ \begin{array}{l}
q(f(X)) \leftarrow q(X), r(X, f(X)) \\
\end{array} \right\},
\]

\[
P_6 = \left\{ \begin{array}{l}
r(f(X), f(Y)) \leftarrow r(X, Y) \\
\end{array} \right\}.
\]

For a surprising fact \( \beta = p(f^3(a)) \), Figure 2 illustrates the applied programs and hypotheses of \( \beta \) for \( \{ P_4, P_5, P_6 \} \). For the applied program \( P_4, P_4 \cup K_1 \vdash \beta \). For the applied programs \( P_4 \) and \( P_5 \), \( P_4 \cup P_5 \cup K_2 \vdash \beta \) and \( P_1 \cup P_5 \cup K_3 \vdash \beta \). For the applied programs \( P_4, P_5, \) and \( P_6, P_4 \cup P_5 \cup P_6 \cup K_4 \vdash \beta \) and \( P_1 \cup P_3 \cup P_6 \cup K_5 \vdash \beta \). The hypothesis \( K_0 \) is a trivial hypothesis, that is, \( K_0 \vdash \beta \).
<table>
<thead>
<tr>
<th>Applied Programs</th>
<th>Hypotheses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td>(K_0 = {p(f(f(a))))} )</td>
</tr>
<tr>
<td>(P_i)</td>
<td>(K_1 = {p(f(a)), q(f(a)))} )</td>
</tr>
<tr>
<td>(P_4, P_5)</td>
<td>(K_2 = {p(f(a)), q(a), r(a, f(a)), r(f(a), f(a)))} )</td>
</tr>
<tr>
<td>(P_4, P_5, P_6)</td>
<td>(K_3 = {p(f(a)), q(f(a)), r(f(a), f(a)))} )</td>
</tr>
</tbody>
</table>

Figure 2: Applied programs and hypotheses \(K_i\) of \(\beta\) for \(\{P_4, P_5, P_6\}\)

Let \(P_i\) \((1 \leq i \leq n)\) be a program. If any program \(P_i\) is given before we apply to rule-finding abduction, then the termination of rule-finding abduction is reduced to one of rule-selecting abduction [Hir93] for the union \(P_1 \cup \cdots \cup P_n\) of programs.

On the other hand, when we discuss the termination of abduction, we can adopt at least two strategies. One is the restriction of class of programs. The discussion of termination of rule-selecting abduction [Hir93] gives an example of it. The other is the introduction of the criterion of termination. For example, in explanation-based generalization [Duv91, HiA94a], it is given as an operationality criterion. In the following sections, we discuss the later strategy for the termination of rule-finding abduction.

In rule-finding abduction, we should choose the programs whose rule-finding abduction terminates. Hence, it is our purpose in the following sections how to choose the programs to avoid an infinite process of rule-finding abduction.

## 4 Abducible Predicate

An abducible predicate (abducible, for short) is defined in the abductive framework [Dun91, EK89, KM90, Poo88]. First, we give the definition of the abductive framework as follows:

**Definition 3** (Poole [Poo88]) An abductive framework is defined as the triple \((P, I, A)\), where \(P\) is a set of Horn clause, \(I\) is an integrity constraint, and \(A\) is a set of predicate symbols called abducible.

Since we only deal with the abductive framework of definite programs, the abductive framework is defined as the pair \((P, A)\) without an integrity constraint, where \(P\) is a definite program. Here, an abducible means the set of predicate symbols of atoms which are assumed true or are hypotheses.

In an abductive framework, an explanation of \(\alpha\) for \((P, A)\) is defined as follows:

**Definition 4** Let \(\alpha\) be a ground atom, and \((P, A)\) be an abductive framework. Then, an explanation of \(\alpha\) for \((P, A)\) is a set \(H\) of atoms such that \(P \cup H \vdash \alpha\) and \(P \cup H \subseteq A\).

**Example 1** Let \(P_7\) be the following program and \(A = \{q\}\).

\[
P_7 = \begin{cases} p(X) \leftarrow q(X) \\ q(X) \leftarrow p(X) \end{cases}.
\]

Then, for a ground atom \(\alpha = p(a)\), the set \(H = \{q(a)\}\) is an explanation of \(\alpha\) for \((P_7, A)\). On the other hand, let \(A' = \{r\}\). Then, there exists no explanation of \(\alpha\) for \((P_7, A')\).
An abducible is similar to an operationality criterion, which is introduced in explanation-based generalization (EBG, for short). Note that the purpose of abductive framework is different from that of EBG. An abductive framework is related to nonmonotonic reasoning or knowledge representation, while EBG is related to machine learning or knowledge acquisition.

For a ground atom \( \alpha \), the leaves of the proof tree of \( \alpha \) given by EBG are elements of an operationality criterion, and we can regard them as abducible. Note that, in EBG, an operationality criterion is given before a proof tree is constructed. In other words, an operationality criterion is introduced in order to guarantee that the proof tree is finite.

Then, which of atoms is an abducible?

If a proof tree is finite, then the leaves of it are possible to be an abducible. Furthermore, for the set \( H \) of nodes in the proof tree, if any branch of the proof tree includes at least one element of \( H \), then \( H \) can be regarded as an abducible.

However, if the class of programs is not restricted, we cannot determine before the proof tree is constructed whether or not the branch of a proof tree is finite. Hence, in the next section, we investigate the syntactical characterization of programs whose proof trees have an infinite branch.

5 Loop-Pair

When we debug a Prolog program, we search for the proof trees of it, and check whether or not it correctly works according as our intention. If there exists an infinite branch of the proof trees, then this program is not designed with our intention. Hence, it is an important view for Prolog debugging to determine whether or not the branch of a proof tree is infinite. In order to solve this problem, we introduce the concept of a loop-pair. We deal with the loop-pair to syntactically characterize the termination of rule-finding abduction.

**Definition 5** Let \( s \) and \( t \) be terms. Then, a loop-pair \( \langle s, t \rangle \) is inductively defined as follows:

1. If \( s \) is a constant symbol \( a \), then \( t \) is a term which includes the constant symbol \( a \) or a variable \( X \) as subterm.
2. If \( s \) is a variable \( X \), then \( t \) is either a term which includes the variable \( X \) as subterm, or a variable \( Y \).
3. If \( s \) is a term \( f(s_1, \ldots, s_m) \), \( t \) is a term \( f(t_1, \ldots, t_m) \), and \( \langle s_i, t_i \rangle \) is a loop-pair for any \( i \) (\( 1 \leq i \leq m \)), then so is \( \langle s, t \rangle \).

**Example 2** The following pairs are loop-pairs:

\[
\langle a, a \rangle, \langle a, f(a) \rangle, \langle a, X \rangle, \langle a, f(X) \rangle \\
\langle X, X \rangle, \langle X, f(X) \rangle, \langle Y, X \rangle \\
\langle f(a, b), f(X, b) \rangle, \langle g(a, f(X)), g(X, f(Y)) \rangle
\]

**Definition 6** Let \( \alpha \) and \( \beta \) be atoms \( p(s_1, \ldots, s_n) \) and \( p(t_1, \ldots, t_n) \), respectively. Then, \( \langle \alpha, \beta \rangle \) is a loop-pair if \( \langle s_i, t_i \rangle \) is a loop-pair for any \( i \) (\( 1 \leq i \leq n \)).

**Lemma 1** Let \( \alpha \) and \( \beta \) be atoms \( p(s_1, \ldots, s_n) \) and \( \beta = p(t_1, \ldots, t_n) \), respectively. Let \( C \) be the following clause:

\[
C = p(u_1, \ldots, u_n) \leftarrow p(v_1, \ldots, v_n).
\]

If \( \langle \alpha, \beta \rangle \) is a loop-pair, \( \alpha \theta = p(u_1, \ldots, u_n) \theta \), and \( \beta = p(v_1, \ldots, v_n) \theta \), then there exists an atom \( \gamma \) such that
1. $\langle\beta, \gamma\rangle$ is a loop-pair,

2. there exists a substitution $\sigma$ such that $\beta \sigma = p(u_1, \cdots, u_n)\sigma$, and

3. $\gamma = p(v_1, \cdots, v_n)\sigma$.

**Proof** Let $\gamma$ be an atom $p(w_1, \cdots, w_n)$. The result is proven by mathematical induction on the structure of $t_i$. Note that different capital letters represent different variables.

1. If $t_i$ is a constant symbol $a$, then $s_i = a$ and $u_i = v_i = X$. Hence, $w_i = a$.

2. If $t_i$ is a variable $X$, then the following three cases hold:
   
   (a) If $s_i$ is a constant symbol $a$, then $u_i = U$ and $v_i = V$. Hence, $w_i = W$.

   (b) If $s_i$ is the variable $X$, then $u_i = v_i = U$. Hence, $w_i = X$.

   (c) If $s_i$ is a variable $Y$ different from $X$, then $u_i = U$ and $v_i = V$. Hence, $w_i = W$.

3. If $t_i$ is the form of $f(t'_1, \cdots, t'_n)$, then the following two cases hold:
   
   (a) If $s_i$ is a subterm of $t_i$, then $u_i$ is also a subterm of $v_i$. Hence, $t_i$ is a subterm of $w_i$.

   (b) Otherwise, $s_i$ is the form of $f(s'_1, \cdots, s'_n)$ and suppose that $\langle\langle s'_i, t'_i\rangle\rangle$ is a loop-pair for any $i$ ($1 \leq i \leq n$). Then, $s'_i \theta = u'_i \theta$, $t'_i = v'_i \theta$, $t'_i \sigma = u'_i \sigma$, and $v'_i = v'_i \sigma$. Hence, $s_i \sigma = u_i \sigma$ and $w_i = v_i \sigma$.

Hence, in each case, $\langle\langle t_i, w_i\rangle\rangle$ is a loop-pair, and $w_i = v_i \sigma$ for some substitution $\sigma$. Therefore, $\langle\langle \beta, \gamma\rangle\rangle$ is a loop-pair, $\beta \sigma = p(u_1, \cdots, u_n)\sigma$, and $\gamma = p(v_1, \cdots, v_n)\sigma$. $\square$

**Theorem 2** Let $P$ be a definite program, $\alpha$ and $\beta$ be atoms, and $C_1, \cdots, C_n \in P$ be the applied clauses in the derivation from the goal $\leftarrow \alpha$ to the goal $\leftarrow \beta$ in $P$. If $\langle\langle \alpha, \beta\rangle\rangle$ is a loop-pair, then there exists an atom $\gamma$ such that

1. $\langle\langle \beta, \gamma\rangle\rangle$ is a loop-pair, and

2. the goal $\leftarrow \gamma$ is derived from $\leftarrow \beta$ by applying the clauses $C_1, \cdots, C_n \in P$.

**Proof** For any $C_i$, by applying the selected atoms in the derivation from $\leftarrow \alpha$ to $\leftarrow \beta$, there exists an atom $\gamma$ which satisfies the above condition 2. Then, we can reduce the result to Lemma 1. $\square$

Let $P$ be a definite program. If all predicate symbols in the head of clauses in $P$ are distinct from each other, then the input clauses in a derivation are determined uniquely for a goal. Hence, the following corollary holds:

**Corollary 1** Let $P$ be a definite program and $\alpha$ be an atom. Suppose that all predicate symbols in the heads of clauses in $P$ are mutually distinct. If a loop-pair appears in the branch of the proof tree of $\alpha$ on $P$, then this branch is infinite.

By Corollary 1, we can select the programs which do not include this branch, in order to avoid infinite branches of the proof tree.
6 Loop-Elimination

In rule-finding abduction, we can deal with the set of programs given in advance. Then, in this section, we discuss the termination of rule-finding abduction by choosing programs.

It is a useful method for Prolog debugging to obtain and to analyze the transformed program whose termination is guaranteed, and to debug the original one. In this section, we discuss the termination of rule-finding abduction from this viewpoint. Hirata has already captured the termination of Prolog programs as head-reducing programs [Hir93]. Hence, this section also begins with head-reducing programs.

For a program $P'$, if a program $P$ is head-reducing with respect to the predicate $p$, is the union $P \cup P'$ head-reducing with respect to the predicate $p$?

**Example 3** Let $P_8$ and $P_9$ be programs \{\begin{align*} p(X) &\leftarrow q(X) \\
q(X) &\leftarrow p(X) \end{align*}\} and \{\begin{align*} q(X) &\leftarrow p(X) \end{align*}\}. Clearly $P_8$ and $P_9$ are head-reducing with respect to the predicate $p$. Then, the union $P_8 \cup P_9$ is the following program:

$$P_8 \cup P_9 = \begin{cases} p(X) &\leftarrow q(X) \\
q(X) &\leftarrow p(X) \end{cases}.$$  

Obviously, $P_8 \cup P_9$ is not head-reducing with respect to the predicate $p$.

In general, even if $P$ and $P'$ are head-reducing with respect to the same predicate, $P \cup P'$ is not always head-reducing with respect to the same predicate. Then, is there the choice of programs whose union is head-reducing? In particular, for a clause $C$ and a head-reducing program $P$ with respect to the predicate $p$, we consider the condition under which $P \cup \{C\}$ is head-reducing with respect to $p$. First, we define *reducing programs*, which are more restricted than head-reducing programs, introduced by Yamamoto [Yam92].

**Definition 7** (Yamamoto [Yam92]) A clause $A \leftarrow B_1, \cdots, B_n$ is reducing if $|A \theta| > |B_i \theta|$ ($1 \leq i \leq n$) for any substitution $\theta$. A program $P$ is a *reducing program* if all clauses in $P$ are reducing.

By Definition 7, any reducing program is also head-reducing with respect to any predicate. Furthermore, if $P$ is a reducing program and $C$ is reducing, then $P \cup \{C\}$ is also a reducing program. Then, $P \cup \{C\}$ is head-reducing with respect to any predicate. However, the following cases 1 and 2 hold:

1. Let $P_{10}$ be a program \{\begin{align*} p(f^2(X)) &\leftarrow p(X), q(f(X)) \end{align*}\}, and $C_{10}$ be a clause $q(X) \leftarrow p(f^3(X))$. Then, $P_{10} \cup \{C_{10}\}$ is not head-reducing with respect to the predicate $p$. Hence, even if $P$ is a reducing program and $C$ is $p$-reducing with respect to all arguments, $P \cup \{C\}$ is not always head-reducing with respect to $p$.

2. Let $P_{11}$ be a program \{\begin{align*} p(X) &\leftarrow q(Y) \end{align*}\} and $C_{11}$ be a clause $q(f(X)) \leftarrow p(X)$. Then, $P_{11} \cup \{C_{11}\}$ is not head-reducing with respect to the predicate $p$. Hence, even if $P$ is head-reducing with respect to the predicate $p$ and $C$ is reducing, $P \cup \{C\}$ is not always head-reducing with respect to $p$.

Hence, if we extend a reducing program $P$ to a head-reducing program with respect to the predicate $p$, or extend a reducing clause $C$ to a $p$-reducing clause with all arguments, then $P \cup \{C\}$ is not always head-reducing with respect to the predicate $p$.

Let $P$ be a head-reducing program with respect to the predicate $p$ and $C$ be a $p$-reducing clause with respect to all arguments. In the remainder of this section, we consider the method to combine $P$ with $C$. Then, it is our purpose to eliminate the infinite branches of proof trees of $P \cup \{C\}$.

First, we introduce the following transformation of a clause $C$ for a program $P$. 

8
Definition 8 Let $P$ be a program and $C$ be a clause $A \leftarrow B_1, \ldots, B_l$. Then, loop-elimination of $C$ for $P$, denoted by $le(C, P)$, is a clause which is replaced the predicate symbol $q$ in $B_i$ ($1 \leq i \leq l$) appearing in some head of $P$ by the predicate $true.q$.

Then, the following lemma holds.

Lemma 2 Let $P$ be a program and $C$ be a clause. If $P$ is head-reducing with respect to the predicate $p$ and $C$ is $pred(head(C))$-reducing with respect to all arguments, then $P \cup \{le(C, P)\}$ is head-reducing with respect to the predicate symbol $p$.

Proof Suppose that $P$ is a program $\{p(t_1, \ldots, t_n) \leftarrow D_1, \ldots, D_m\}$ and $C$ is a clause $A \leftarrow B_1, \ldots, B_l$.

If any $D_j$ and $A$ are not unifiable, then the result trivially holds. Suppose $D_j \theta = A \theta$. Let $le(C, P)$ be loop-elimination $A \leftarrow B'_1, \ldots, B'_l$ of $C$ for $P$. Then, $rec(P \cup \{le(C, P)\}, p)$ is constructed in the following way:

$$p(t_1, \ldots, t_n) \leftarrow D_1 \theta, \ldots, D_{j-1} \theta, (B'_1 \theta, \ldots, B'_l \theta), D_{j+1} \theta, \ldots, D_m \theta.$$ 

Then, $RP(P \cup \{le(C, P)\}, p) = \{rec(P \cup \{le(C, P)\}, p), le(C, P)\}$. By Definition 8, the predicate $p$ appearing in the bodies of $le(C, P)$ is replaced by $true.p$. Then, the predicate $p$ does not appear in the part $(B'_1 \theta, \ldots, B'_l \theta)$ in the body of $rec(P \cup \{le(C, P)\}, p)$. Furthermore, since $C$ is $pred(A)$-reducing with respect to all arguments, for $B_i$ such that $pred(A) = pred(B'_i)$, $|s_k \theta| > |t_k \theta|$ for any argument’s term $t_k$ and $s_k$ ($1 \leq k \leq n$) of $A$ and $B'_i$. For $B'_i$ such that $pred(A) \neq pred(B'_i)$, any definition clauses of $pred(B'_i)$ does not appear in $RP(P \cup \{le(C, P)\}, p)$. Hence, $RP(P \cup \{le(C, P)\}, p)$ is head-reducing with respect to $p$. □

If a clause $C$ is not $pred(A)$-reducing with respect to all arguments, then there exists the following counterexample of Lemma 2.

Let $P_{12}$ be a program $\{p(f(X)) \leftarrow q(X, Y)\}$ and $C_{12}$ be a clause $q(X, f(Y)) \leftarrow q(f(X), Y)$. Then, $P_{12}$ is head-reducing with respect to the predicate $p$, but $C_{12}$ is not $q$-reducing with respect to the first argument. Note that $le(C_{12}, P_{12})$ is equal to $C_{12}$ itself. Then, for the goal $\leftarrow p(f^2(a))$, the derivations of $P_{12} \cup \{C_{12}\} \cup \{\leftarrow p(f^2(a))\}$ are infinite.

The next theorem claims that, by loop-elimination, we can choose the several programs whose union is head-reducing. In other words, rule-finding abduction for $P \cup \{le(C, P)\}$ terminates.

Theorem 3 Let $P$ be a program and $C$ be a clause. If $P$ is head-reducing with respect to the predicate $p$, $P$ includes the definition clause of $p$, and $C$ is $pred(head(C))$-reducing with respect to all arguments, then $P \cup \{le(C, P)\}$ is also head-reducing with respect to the predicate symbol $p$.

Proof The result is proven by mathematical induction on the number $|P|$ of clauses in $P$. If the number is 1, that is, $|P| = 1$, then Lemma 2 implies the result.

Suppose that the result is true for $|P| = n$, and consider the result for $|P| = n + 1$. Let $P$ be a program $\{C_1, \ldots, C_n\}$ and $P'$ be a program $P \cup \{C_{n+1}\}$. Let $C, le(C, P)$ and $le(C, P')$ be the following clauses:

$$C = A \leftarrow B_1, \ldots, B_l,$$
$$le(C, P) = A \leftarrow B'_1, \ldots, B'_l,$$
$$le(C, P') = A \leftarrow B''_1, \ldots, B''_l.$$
By the induction hypothesis, all of programs \( P, P' \cup \{ e(C, P') \} \), and \( P' \) are head-reducing with respect to \( p \). If \( C_{n+1} \) is not applied to the construction of \( rec(P', p) \), then \( rec(P', p) = rec(P, p) \), and the result holds by the induction hypothesis.

Suppose that \( C_{n+1} \) is applied to the construction of \( rec(P', p) \). Then, there exists an index \( i \) such that \( head(C_{n+1}) \) is unifiable with an atom in the body of \( C_i \). Let \( C_i \) and \( C_{n+1} \) be the following clauses:

\[
C_i = D \leftarrow E_1, \ldots, E_j, \ldots, E_m, \\
C_{n+1} = F \leftarrow G_1, \ldots, G_k, \ldots, G_f.
\]

Then, the recursive definition \( rec(P' \cup \{ e(C, P') \}, p) \) is constructed in the following way: Suppose that the following clause is an intermediate clause in constructing the recursive definition:

\[
p(t_1, \ldots, t_h) \leftarrow A_1, \ldots, D', \ldots, A_g.
\]

Since \( C_{n+1} \) is applied to the construction of \( rec(P', p) \), suppose that \( D' \sigma = D \sigma \). Then, by application of \( C_i \), we obtain the following clause:

\[
p(t_1, \ldots, t_h) \sigma = A_1 \sigma, \ldots, (E_1 \sigma, \ldots, E_j \sigma, \ldots, E_m \sigma), \ldots, A_g \sigma.
\]

Since \( head(C_{n+1}) \) is unifiable with an atom in the body of \( C_i \), suppose that \( E_j \sigma \theta = F \theta \). Then, by application of \( C_{n+1} \), we also obtain the following clause:

\[
p(t_1, \ldots, t_h) \sigma \theta = A_1 \sigma \theta, \ldots, (E_1 \sigma \theta, \ldots, E_j \sigma \theta, \ldots, E_m \sigma \theta), \ldots, A_g \sigma \theta.
\]

For any index \( k (1 \leq k \leq f) \), if \( G_k \theta \) and \( A \) are not unifiable, then, by the induction hypothesis, \( P \cup \{ le(C, P') \} \) is head-reducing with respect to \( p \). Hence, \( P' \cup \{ le(C, P') \} \) is also head-reducing with respect to \( p \).

Otherwise, suppose that there exists a unifier \( \lambda \) for \( G_k \theta \) and \( A \). Then, \( G_k \theta \lambda = A\lambda \). The recursive definition \( rec(P' \cup \{ le(C, P') \}, p) \) is also constructed in the following way:

\[
p(t_1, \ldots, t_h) \sigma \lambda = A_1 \sigma \lambda, \ldots, (E_1 \sigma \lambda, \ldots, (G_1 \theta \lambda, \ldots, (B_1'' \lambda, \ldots, B_f'' \lambda), \ldots, G_f \theta \lambda, \ldots), \ldots, E_m \sigma \lambda), \ldots, A_g \sigma \theta.
\]

By the definition of \( le(C, P') \) and by the construction of \( rec(P' \cup \{ le(C, P') \}, p) \), \( B_i'' \) and any head of the clauses in \( P' \) are not unifiable. Consequently, for some substitution \( \mu \), the recursive definition \( rec(P' \cup \{ le(C, P') \}, p) \) is constructed as follows:

\[
p(t_1, \ldots, t_h) \mu = H_1 \mu, \ldots, (B_1'' \mu, \ldots, B_f'' \mu), \ldots, H_n \mu.
\]

Since \( P' \) is head-reducing with respect to \( p \) by the induction hypothesis, an atom \( H_i \mu \) except \( B_i'' \mu \) (\( 1 \leq i \leq l \)) satisfies the conditions that \( RP(P' \cup \{ le(C, P') \}, p) \) is head-reducing with respect to \( p \). Furthermore, for any atom \( B_i'' \mu \) (\( 1 \leq i \leq l \)), the definition clause of \( pred(B_i'') \) does not appear in \( P' \) by the definition of \( le(C, P'') \). On the other hand, \( C \), so \( le(C, P'') \), is \( pred(A) \)-reducing with respect to all arguments. Hence, \( RP(P' \cup \{ le(C, P') \}, p) \), which includes \( rec(P' \cup \{ le(C, P') \}, p) \), is head-reducing with respect to \( p \).

Therefore, \( P' \cup \{ le(C, P') \} \) is head-reducing with respect to \( p \). \( \square \)

The loop-elimination of \( P' \) for \( P \), denoted by \( le(P', P) \), is the set of \( le(C, P) \), where \( C \) is a clause in \( P' \). In other words,

\[
le(P', P) = \{ le(C, P) \mid C \in P' \}.
\]
By Theorem 3, when the programs $P$ and $P'$ are given, rule-finding abduction for $P \cup le(P', P)$ also terminates.

**Example 4** Let $P_{13}$ and $P_{14}$ be the following programs:

$$P_{13} = \left\{ \begin{array}{l} p(f(X)) \leftarrow q(X) \\ q(X) \leftarrow p(X) \end{array} \right\},$$

$$P_{14} = \left\{ \begin{array}{l} p(X) \leftarrow q(X) \\ q(X) \leftarrow r(X), s(X) \end{array} \right\}.$$

Since the union $P_{13} \cup P_{14}$ is not head-reducing with respect to the predicate $p$ nor $q$, this program falls into an infinite loop for any ground goal with the predicates $p$ and $q$.

On the other hand, after transforming $P_{14}$ to loop-elimination $le(P_{14}, P_{13})$ of $P_{14}$ for $P_{13}$, then we obtain the following set of clauses:

$$le(P_{14}, P_{13}) = \left\{ \begin{array}{l} p(f(X)) \leftarrow q(X) \\ q(X) \leftarrow p(X) \\ p(X) \leftarrow true, q(X) \\ q(X) \leftarrow r(X), s(X) \end{array} \right\}.$$

Furthermore, after transforming $P_{13}$ to loop-elimination $le(P_{13}, P_{14})$ of $P_{13}$ for $P_{14}$, we also obtain the following set of clauses:

$$le(P_{13}, P_{14}) = \left\{ \begin{array}{l} p(f(X)) \leftarrow q(X) \\ q(X) \leftarrow true, p(X) \\ p(X) \leftarrow q(X) \\ q(X) \leftarrow r(X), s(X) \end{array} \right\}.$$

By Theorem 3, rule-finding abduction for both $le(P_{14}, P_{13})$ and $le(P_{13}, P_{14})$ terminate for any ground goal.

7 Analogy Reasoning

In computer science, there are various researches for analogy reasoning. In this paper, we adopt the analogy reasoning introduced by Haraguchi and Arikawa [Har85, HaA86, HiA94b].

Haraguchi and Arikawa [Har85, HaA86, HiA94b] formulated analogy reasoning for logic programming, and defined a formal analogy as the relation between elements in Herbrand universes. In this section, we prepare some concepts on analogy reasoning necessary for our later discussion.

Let $P_b$ and $P_t$ be programs. The program $P_b$ is called a base program and $P_t$ a target program. Then, a finite set $\varphi \subseteq U(P_b) \times U(P_t)$ is called a pairing, where $U(P_b)$ and $U(P_t)$ are Herbrand universes for $P_b$ and $P_t$, respectively. We assume implicitly that $U(P_b) \cap U(P_t) \neq \emptyset$.

**Definition 9** Let $\varphi \subseteq U(P_b) \times U(P_t)$ be a pairing. The set $\varphi^+ \subseteq U(P_b) \times U(P_t)$ is defined to be the smallest set that satisfies the following conditions:

1. $\varphi \subseteq \varphi^+$.
2. If $(l_1, s_1), \ldots, (l_n, s_n) \in \varphi^+$, then $(f(l_1, \ldots, l_n), f(s_1, \ldots, s_n)) \in \varphi^+$.

**Definition 10** Let $\alpha$ and $\beta$ be ground atoms, and $\varphi \subseteq U(P_b) \times U(P_t)$ be a pairing. Then, $\alpha$ and $\beta$ are identical by $\varphi$, denoted by $\alpha \varphi \beta$, if $\alpha$, $\beta$, and $\varphi$ satisfy the following condition:
\[ \alpha = p(t_1, \ldots, t_n), \]
\[ \beta = p(s_1, \ldots, s_n), \]
\[ \langle t_i, s_i \rangle \in \varphi^+ \quad (1 \leq i \leq n) \]

**Definition 11** Let \( \varphi \subseteq U(P_b) \times U(P_t) \) be a pairing. Then, \( \varphi \) is a *partial identity* between \( P_b \) and \( P_t \) if \( \varphi^+ \) is a one-to-one relation.

In order to formulate abduction and analogy in the same framework, we introduce the following notations: Let \( \varphi \subseteq U(P_b) \times U(P_t) \) be a partial identity between \( P_b \) and \( P_t \).

1. Let \( t \) and \( s \) be terms in \( P_b \) and \( P_t \), respectively. Then, \( t\varphi \) is a term which is obtained by replacing any term \( t' \) in \( t \) such that \( \langle t', s \rangle \in \varphi \) with a term \( s' \). Similarly, \( \varphi s \) is a term which is obtained by replacing any term \( s' \) in \( s \) such that \( \langle t', s' \rangle \in \varphi \) with a term \( t' \).

2. Let \( \alpha = p(t_1, \ldots, t_n) \) and \( \beta = p(s_1, \ldots, s_m) \) be atoms in \( P_b \) and \( P_t \), respectively. Then, atoms \( \alpha \varphi \) and \( \varphi \beta \) are defined as follows:

\[ \alpha \varphi = p(t_1 \varphi, \ldots, t_n \varphi), \]
\[ \varphi \beta = p(\varphi s_1, \ldots, \varphi s_m). \]

3. Let \( C = A \leftarrow A_1, \ldots, A_n \) and \( D = B \leftarrow B_1, \ldots, B_m \) be clauses in \( P_b \) and in \( P_t \), respectively. Then, clauses \( C \varphi \) and \( \varphi D \) are defined as follows:

\[ C \varphi = A \varphi \leftarrow A_1 \varphi, \ldots, A_n \varphi, \]
\[ \varphi D = \varphi B \leftarrow \varphi B_1, \ldots, \varphi B_m. \]

4. Let \( P_b = \{C_1, \ldots, C_n\} \) and \( P_t = \{D_1, \ldots, D_m\} \). Then, programs \( P_b \varphi \) and \( \varphi P_t \) are respectively defined as follows:

\[ P_b \varphi = \{C_1 \varphi, \ldots, C_n \varphi\}, \]
\[ \varphi P_t = \{\varphi D_1, \ldots, \varphi D_m\}. \]

Hirowatari and Arikawa [HiA94b] have introduced the concept of a *partially isomorphic generalization*, which is a generalization of one atom and is the useful tool for analogical reasoning. In this section, we prepare the notions for partially isomorphic generalization.

Let \( \alpha \) be an atom. A term \( t \) is a replaceable term of \( \alpha \) if \( t \) is a constant symbol or a term \( f(X_1, \ldots, X_n) \), where \( f \) is a function symbol and each \( X_i \) is a variable which does not appear in the other terms in \( \alpha \). For a replaceable term \( t \) of \( \alpha \), let \( \alpha[t] \) be an atom obtained by replacing each \( t \) in \( \alpha \) by a new variable \( Z \) which does not appear in \( \alpha \). Then, we write \( \alpha \rightarrow^* \beta \) when \( \alpha[t] \) is a variant of \( \beta \). We define \( \rightarrow^* \) as the reflexive and transitive closure of \( \rightarrow \).

**Definition 12** (Hirowatari and Arikawa [HiA94b]) Let \( \alpha \) and \( \beta \) be atoms. Then, \( \beta \) is a *partially isomorphic generalization* of \( \alpha \) if \( \alpha \rightarrow^* \beta \).

For a set of atoms \( S \), let \([S]\) denote the equivalence class of all atoms in \( S \). In particular, for any \( \alpha \in [S] \) and \( \beta \in [S] \), \( \alpha \) is a variant of \( \beta \).

We can develop analogical reasoning [Har85, HaA86] by the notions of partially isomorphic generalization. Hirowatari and Arikawa [HiA94b] has shown the following three theorems.
Theorem 4 (Hirowatari and Arikawa [HiA94b]) Let $\alpha$ be an atom and $S$ be the set of all partially isomorphic generalizations of $\alpha$. Then, $|S|$ is a lattice whose partial order is $\rightarrow^*$, meet operator is the greatest instantiation, and join operator is the least generalization.

Theorem 5 (Hirowatari and Arikawa [HiA94b]) Let $\alpha$ be a ground atom $p(t_1, \cdots, t_n)$ and $k = |t_1| + \cdots + |t_n|$. Then, a partially isomorphic generalization of $\alpha$ can be computed in $O(k^2)$ time.

Theorem 6 (Hirowatari and Arikawa [HiA94b]) Let $\alpha$ and $\beta$ be ground atoms in $P_b$ and $P_t$, respectively, and $\alpha'$ be the greatest partially isomorphic generalization of $\alpha$. If there exists a substitution $\theta$ such that $\alpha' = \beta\theta$, then there exists an analogy $\varphi \subseteq U(P_b) \times U(P_t)$ such that $\alpha \varphi \beta$.

Here, an analogy $\varphi$ is regarded as a partial function from $U(P_b)$ to $U(P_t)$. By partially isomorphic generalizations, we can obtain the analogy which is guaranteed one direction of partial identity.

8 Rule-Finding Abduction with Analogy

In the previous sections, we have discussed rule-finding abduction. In these discussions, we assumed that the found rules are not ground. If all of the rules in programs are ground, then we cannot apply rule-finding abduction to them, because the application of rule-finding abduction is based on the unification. Consider the following example.

Example 5 Let $p(a)$ be a surprising fact, and $P_{15}$ and $P_{16}$ be the following programs:

\[
\begin{align*}
P_{15} &= \emptyset, \\
P_{16} &= \{ p(b) \leftarrow q(b) \}.
\end{align*}
\]

By rule-finding abduction for $P_{15} \cup P_{16}$, we can propose only a trivial hypothesis $\{ p(a) \}$ of $p(a)$.

In Example 5, if we introduce the analogy such that $a$ is analogous to $b$, then we can obtain a hypothesis $\{ q(b) \}$. Hence, in this section, we discuss such a rule-finding abduction and analogy in the same framework.

Thargad [Tha88] and Duval [Duv91] have tried to discuss abduction and analogy in the same framework. Thagard [Tha88] has applied Kuhn’s philosophy of science [Kuh70] to computer science, and dealt with analogical abduction, which is one of the methods of discovery. On the other hand, Duval [Duv91] has also dealt with abduction and analogy in the framework of explanation-based generalization. However, in such researches, the relationship between abduction and analogy are not clear, since their concepts of abduction and analogy are ambiguous.

In this paper, we adopt the formulation of analogical reasoning by Haraguchi and Arikawa [Har85, HaA86, HiA94a, HiA94b]. It is based on the analogy between Herbrand universes of a base program $P_b$ and that of a target program $P_t$.

Let $P_b$ be a base program, $P_t$ be a target program, and $\alpha$ be a ground atom. Then, a proof tree of $\alpha$ for $P_b$ (resp., $P_t$) is denoted by $T^b_\alpha$ (resp., $T^t_\alpha$). The leaves of $T^b_\alpha$ (resp., $T^t_\alpha$) is denoted by $\text{leaves}(T^b_\alpha)$ (resp., $\text{leaves}(T^t_\alpha)$).

In the formulation of analogical reasoning [Har85, HaA86, HiA94a, HiA94b], it is natural to consider that a surprising fact is given in a target program $P_t$, not in a base program $P_b$. Hence, in the definitions in this section, a surprising fact is also given in a target program $P_t$. 

13
Figure 3: Deducible hypothesis, where \( P_b \vdash \varphi \beta \)

Now, we discuss abduction and analogy in the same framework. In this section, it is our purpose to formulate rule-finding abduction incorporating with analogical reasoning. In other words, we deal with the concept of analogy in order to extend rule-finding abduction. Such the abduction is called rule-finding abduction with analogy.

First, we formulate a simple hypothesis of rule-finding abduction with analogy as follows:

**Definition 13** Let \( P_b = R_b \cup F_b \) and \( P_t \) be programs, and \( \alpha \) be a surprising fact with respect to \( P_t \). Let \( \varphi \subseteq U(P_b) \times U(P_t \cup \{ \alpha \}) \) be a partial identity. Then, a set \( H \) of atoms is a *simple hypothesis* of \( \alpha \) for \( P_b \) and \( P_t \) with analogy \( \varphi \) if \( H \) satisfies the following condition:

\[
R_b \varphi \cup P_t \cup H \vdash \alpha.
\]

However, for the simple hypothesis, the variables in \( P_b \) or \( P_t \) are substituted by ground terms in \( P_b \cup P_t \). In order to solve this problem, we introduce another formulation of abduction with analogy, called a *deducible hypothesis*, as follows:

**Definition 14** Let \( \alpha \) be a surprising fact, that is, \( P_t \not\vdash \alpha \), and \( \varphi \subseteq U(P_b) \times U(P_t \cup \{ \alpha \}) \) be a partial identity. Suppose that \( P_t \cup H_0 \vdash \alpha \). For any \( \beta \in H_0 \), if \( P_b \vdash \varphi \beta \), then \( H = (H_0 - \{ \beta \}) \cup H_1 \varphi \), where \( H_1 \) is the set of nodes in \( T_{\varphi \beta}^1 \). Then, \( H \) is called a *deducible hypothesis* of \( \alpha \) for \( P_b \) and \( P_t \) with analogy \( \varphi \).

Figure 3 illustrates the formulation of a deducible hypothesis. It is clear that, if \( H \) is a deducible hypothesis, then it is a simple hypothesis. Furthermore, the variables in \( P_b \) (resp., \( P_t \)) are substituted by only ground terms in \( P_b \) (resp., \( P_t \)).

It arises a problem how to choose the deducible hypothesis \( H \). For a proof tree, if we choose any combination of nodes in \( T_{\varphi \beta}^b \), then it is very difficult to obtain all deducible hypotheses. Even if we choose two sets of nodes in \( T_{\varphi \beta}^b \), after the above procedure to obtain deducible hypotheses in \( n \) times, the number of deducible hypotheses is at most \( 2^n \). Hence, for a proof tree, we adopt the choice of only one hypothesis in \( T_{\varphi \beta}^b \).
**deducible hypothesis** $H_{\text{leaves}}$ /* based on the leaves of the proof tree */

**input**: $P_b, P_i$: programs,
$\alpha$: a ground atom,
$\varphi \subseteq U(P_b) \times U(P_i \cup \{\alpha\})$: a partial identity

**output**: a deducible hypothesis $H_{\text{leaves}}$

\[
H := \text{leaves}(T^i_\alpha);
\]

while there exists a ground atom $\beta \in H$ such that $P_b \vdash \varphi \beta$
\[
H_1 := \text{leaves}(T^i_{\beta';\beta});
H' := H_1\varphi;
H'' := \phi;
\]

while $H' = \phi$
\[
\text{choose } \gamma \in H';
H'' := H'' \cup \text{leaves}(T^i_\gamma);
H' := H' - \{\gamma\};
\]

end

$H_{\text{leaves}} := (H - \{\beta\}) \cup H''$

end

output $H_{\text{leaves}}$

---

**Figure 4**: Algorithm to construct a deducible hypothesis $H_{\text{leaves}}$

In order to construct a deducible hypothesis concretely, we introduce a deducible hypothesis $H_{\text{leaves}}$ which is based on the leaves of a proof tree. A deducible hypothesis $H_{\text{leaves}}$ is obtained by the algorithm in Figure 4. In this algorithm, $H_0$ and $H_1$ in Figure 3 are the leaves of $T^i_\alpha$ and $T^i_{\beta';\beta}$, respectively.

Let $P_b$ and $P_i$ be programs, and $\alpha$ be a ground atom. By $P^b_\alpha$ (resp., $P^i_\alpha$), we denote the set of clauses which are applied to a proof tree $T^i_\alpha$ (resp., $T^i_{\beta';\beta}$) of $\alpha$ in $P_b$ (resp., $P_i$). Then, the following theorem holds:

**Theorem 7** Let $P_b$ and $P_i$ be programs, and $\alpha$ be a ground atom. Let $\varphi \subseteq U(P_b) \times U(P_i \cup \{\alpha\})$ be a partial identity and $\text{leaves}(T^i_\alpha)$ be a set $\{\beta_j \mid 1 \leq j \leq k\}$ of leaves in $T^i_\alpha$. Suppose that $P_i \nvdash \alpha$. If $P_b \vdash \varphi \beta_j$ (1 $\leq j \leq k$), then $\{\bigcup_{j=1}^{k} P^b_{\varphi \beta_j} \varphi\} \cup P^i_{\alpha} \vdash \alpha$.

**Proof** For $\beta_j$, $H_{\text{leaves}}$ denotes the deducible hypothesis of $\beta_j$. Then, $H'_{\text{leaves}} \subseteq P^b_{\varphi \beta_j} \varphi$, and the clauses in $P^b_{\varphi \beta_j} \varphi$ are applied to $P_i$. Hence,

\[
P^b_{\varphi \beta_j} \varphi \cup P^i_{\alpha} \cup \text{leaves}(T^i_\alpha) - \{\beta_j\} \vdash \alpha.
\]

By applying the above consideration to $\beta_j$ for $1 \leq j \leq k$, we can obtain the result. □

By Theorem 7, a deducible hypothesis is correct in the sense of analogical reasoning.

In this formulation, we assume that a partial identity $\varphi$ is given in advance. This assumption is unreasonable. In analogical reasoning, an analogy $\varphi$ is not given in advance, and it is a main problem to detect the $\varphi$. Then, in order to obtain an analogy $\varphi$ while constructing $H_{\text{leaves}}$, we adopt the concept of partially isomorphic generalizations, which has been introduced by Hirowatari and Arikawa [HiA94b]. They have regarded an analogy as a function from $U(P_b)$
to $U(P \cup \{\alpha\})$, not a partial identity. They have also reduced the problem of the detection of partial identity to the unification of partially isomorphic generalization as Theorem 6. In rule-finding abduction with analogy, we also follow this consideration.

Consider the following examples.

**Example 6** Let $P_{17}$ be the following base program:

$$
P_{17} = \left\{ \begin{array}{l}
  C_1 : p(a, b) \\
  C_2 : p(f(X), b) \leftarrow p(X, b)
\end{array} \right\}.
$$

Let $\alpha$ be a surprising fact $p(f^2(c), f^2(d))$ with respect to an empty target program.

1. The partially isomorphic generalization $PC_1$ of $C_1$ is as follows:

$$
PC_1 : p(X, Y).
$$

Since $head(PC_1)$ and $\alpha$ are unifiable, and $p(a, b)$ is provable in $P_b$, we obtain the following deducible hypothesis $H_1$:

$$
H_1 = \{p(f^2(c), f^2(d))\}.
$$

There exist substitutions $\theta_1 = \{X := a, Y := b\}$ and $\theta_2 = \{X := f^2(c), Y := f^2(d)\}$ such that $head(PC_1) \theta_1 = p(a, b)$, and $head(PC_1) \theta_2 = \alpha$. Then, by Theorem 6, there exists the analogy $\varphi_1 \subseteq U(P_{17} \times U(\{\alpha\})$ which is obtained by:

$$
\varphi_1 = \{(t, s) \mid X := t \in \theta_1, X := s \in \theta_2\}.
$$

Hence, the analogy $\varphi_1$ for $H_1$ is a set $\{(a, f^2(c)), (b, f^2(d))\}$.

2. The partially isomorphic generalization $PC_2$ of $C_2$ is as follows:

$$
PC_2 : p(f(X), Y) \leftarrow p(X, Y).
$$

Since $head(PC_2)$ and $\alpha$ are unifiable, we obtain the following candidate $K_1$ of deducible hypotheses:

$$
K_1 = \{p(f(c), f^2(d))\}.
$$

For an element $p(f(c), f^2(d))$ of $K_1$, we also continue the above discussion 1 and 2. Then, from $K_1$, we obtain the following deducible hypothesis $H_2$ and the analogy $\varphi_2 \subseteq U(P_{17}) \times U(\{\alpha\})$:

$$
H_2 = \{p(f(c), f^2(d))\}, \varphi_2 = \{(a, f(c)), (b, f^2(d))\}.
$$

Also we obtain the following candidate $K_2$ of deducible hypotheses:

$$
K_2 = \{p(c, f^2(d))\}.
$$

For an element $p(c, f^2(d))$ of $K_2$, $head(PC_1)$ and $p(c, f^2(d))$ are unifiable, while $head(PC_2)$ and $p(c, f^2(d))$ are not. Furthermore, $p(a, b)$ is provable in $P_i$. Then, from $K_2$, we obtain the following deducible hypothesis $H_3$ and the analogy $\varphi_3 \subseteq U(P_{17}) \times U(\{\alpha\})$:

$$
H_3 = \{p(c, f^2(d))\}, \varphi_3 = \{(a, c), (b, f^2(d))\}.
$$
\[
\begin{array}{ccc}
p(a, b) & p(f(a), b) & p(f^2(a), b) \\
H_1 \\ p(a, b) & p(f(a), b) & p(f^2(a), b) \\
H_2 \\ p(a, b) \\
H_3 \\
\varphi_1 \\
\langle a, f^2(c) \rangle & \langle a, f(c) \rangle & \langle a, c \rangle \\
\varphi_2 \\
\langle b, f^2(d) \rangle & \langle b, f^2(d) \rangle & \langle b, f^2(d) \rangle \\
\varphi_3 \\
\end{array}
\]

Figure 5: Deducible hypotheses \(H_1, H_2, \) and \(H_3\), and corresponding proof trees

Figure 5 illustrates three deducible hypotheses \(H_1, H_2, \) and \(H_3\), and proof trees which are corresponding to \(H_i\). For each proof tree, the root node is analogous to a surprising fact \(a\) under the analogy \(\varphi_i\), and the atom which is analogous to leaf node under \(\varphi_i\) is corresponding to a deducible hypothesis \(H_i\).

**Example 7** For \(P_{17}\), if a surprising fact is a ground atom \(p(f^2(c), c)\), then we obtain the following deducible hypotheses \(H_j\) and analogies \(\varphi_j\) (\(4 \leq j \leq 6\)):

- \(H_4 = \{p(a, b)\}\), \(\varphi_4 = \{\langle a, f^2(c) \rangle, \langle b, c \rangle\}\)
- \(H_5 = \{p(f(a), b)\}\), \(\varphi_5 = \{\langle a, f(c) \rangle, \langle b, c \rangle\}\)
- \(H_6 = \{p(f^2(a), b)\}\), \(\varphi_6 = \{\langle a, c \rangle, \langle b, c \rangle\}\)

Note that \(\varphi_j\) (\(4 \leq j \leq 6\)) is a function from \(U(P_{17})\) to \(U(\{\alpha\})\).

On the other hand, let \(P_{18}\) be the following base program:

\[
P_{18} = \left\{ \begin{array}{l}
C_3 : \quad p(a, a) \\
C_4 : \quad p(f(X), a) \leftarrow p(X, a)
\end{array} \right\}
\]

If either \(p(f^2(c), f(d))\) or \(p(f^2(c), d)\) is given as a surprising fact \(\beta\), then there exist no deducible hypotheses. Because we regard an analogy \(\varphi\) as a function from \(U(P_{18})\) to \(U(\{\beta\})\), and, for the above surprising facts and the base program \(P_{18}\), there exist no such the analogies. On the other hand, let \(p(f^2(c), c)\) be a surprising fact. Then, we obtain the following deducible hypothesis \(H_7\) and the analogy \(\varphi_7\):

\[
H_7 = \{p(c, c)\}, \ \varphi_7 = \{\langle a, c \rangle\}
\]

We can realize rule-finding abduction with analogy as the Prolog program in Figure 6. The program **ab.anal** computes a deducible hypothesis \(H_{\text{leaves}}\) and an analogy. Note that, this program assumes that a target program is empty, that is, only the first while-loop in Figure 4 is realized.

The predicate **ab.anal** in Figure 6 is given a surprising fact as its first argument. Then, it returns a deducible hypothesis as its second argument, a pairing as its third argument, and a world as its forth argument for a surprising fact as its first argument. The predicate **analogy** returns the pairing as the third argument between the base rule given as the second argument and the target rule given as the first argument. The predicate **provable** checks provability in a base program \(P_0\), and, if so, then it proposes a deducible hypothesis as the third argument.
ab_analogy(TG, TG, Pair, WorldTarget):-
  functor(TG, Pred, Arity), functor(BG, Pred, Arity),
  world(WorldBase, WorldTarget),
  fact(WorldBase, (BG:-true)),
  analogy((TG:-true), (BG:-true), Pair).

ab_analogy(TG, TGs, Pair, WorldTarget):-
  functor(TG, Pred, Arity), functor(BG, Pred, Arity),
  provable(TG, BG, TGs, BGs, WorldTarget),
  not TG==TGs,
  analogy((TG:-TGs), (BG:-BGs), Pair).

provable(TG, BG, TL, BL, WorldTarget) :-
  rule(TG, BG, TGs, BGs, WorldTarget),
  provable(TGs, BGs, TL, BL, WorldTarget).

provable((TG, TGs), (BG, BGs), (TL, TLs), (BL, BLs), WorldTarget):-
  provable(TG, BG, TL, BL, WorldTarget),
  provable(TGs, BGs, TLs, BLs, WorldTarget).

provable(TG, BG, TG, BG, WorldTarget):-
  world(WorldBase, WorldTarget), fact(WorldBase, (BG:-true)), !.

rule(TG, BG, TGs, BGs, WorldTarget) :-
  world(WorldBase, WorldTarget),
  fact(WorldBase, (BG:-BGs)),
  not BGs=true,
  pig_rule((BG:-BGs), (PG:-PGs)),
  copy((PG:-PGs), (TG:-TGs)).

Figure 6: Program ab_analogy
The predicate \texttt{pig\_rule} returns the partially isomorphic generalization \( PG :- PGs \) of the rule \( BG :- BGs \) as the second argument.

For a clause \( C \), by \( \texttt{pig}(C) \), we denote the partially isomorphic generalization of \( C \). For a program \( P \), by \( \texttt{pig}(P) \), we denote the set \( \{ \texttt{pig}(C) \mid C \in P \} \). The termination of the Prolog program \texttt{ab\_ana}, which is rule-finding abduction with analogy, is characterized in the following theorem as the corollary of Theorem 1.

**Corollary 2** Let \( P_b = R_b \cup P_b \) and \( P_l \) be programs and \( p \) be a predicate symbol. If \( \texttt{pig}(R_b) \cup P_l \) is head-reducing with respect to the predicate \( p \), then all the derivations of \( \texttt{pig}(R_b) \cup P_l \cup \{ \leftarrow p(s_1, \ldots, s_n) \} \) are finite for any ground atom \( p(s_1, \ldots, s_n) \).

By Corollary 2 and Theorem 6, if \( \texttt{pig}(R_b) \cup P_l \) is head-reducing with respect to the predicate \( p \), then, for any surprising fact \( p(s_1, \ldots, s_n) \), the goal

\[
?- \texttt{ab\_ana}(p(s_1, \ldots, s_n), X, P, W)
\]

terminates, and returns the deducible hypotheses as its second argument and the pairings as its third argument.

For the program in Figure 6, if the proof tree of a target program is obtained, then the computational complexity to obtain the deducible hypothesis is characterized as the following theorem:

**Theorem 8** Let \( P_b \) be a base program and \( \alpha \) be a ground atom. For a given proof tree, a ground atom \( \alpha' \) is a root and \( H' \) is the set of leaves. Suppose that \( |\alpha| = k \) and \( |H'| = l \). Then, a deducible hypothesis is computed in \( O(k^3l) \) time.

**Proof** By Theorem 5, for a root \( \alpha' \), the partially isomorphic generalization \( \beta \) of \( \alpha' \) is computed in \( O(k^3) \). Since \( \alpha \) is ground, whether or not \( \beta \) and \( \alpha \) are unifiable is determined in \( O(k) \). If \( \beta \) and \( \alpha \) are unifiable, then, by Theorem 6, an analogy \( \varphi \) can be computed simultaneously. The time complexity to apply this \( \varphi \) to \( H' \) is \( O(l) \). Hence, a deducible hypothesis \( H'\varphi \) is computed in \( O(k^3l) \) time. \( \square \)

The base program in Example 6 is represented as follows:

\begin{verbatim}
fact(w1, p(a,b):-true). fact(w1, p(f(X),b):-p(X,b)). world(w1,w2).
\end{verbatim}

Here, the atom \texttt{world(w1,w2)} represents that \texttt{w1} is a base program and \texttt{w2} is a target program. In this program, a target program \texttt{w2} is empty. Then, we obtain the following results:

\begin{verbatim}
: ?- ab\_ana(p(f(c)),f(f(d))),X,P,W).
X = p(f(f(c)),f(f(d))),
P = [a--f(c),b--f(d)]],
W = w2 ;
X = p(c,f(d))),
P = [a--c,b--f(d)]],
W = w2 ;
X = p(f(c),f(f(d))),
P = [a--c,b--f(d)]],
W = w2 ;
no
: ?- ab\_ana(p(c,d),X,P,W).
\end{verbatim}
\[
X = p(c,d),
P = [a\leftarrow c,b\leftarrow d],
W = w_2;
\]

Note that the solution \([a\leftarrow f(f(c)),b\leftarrow f(f(d))]\) of the third argument for the program \texttt{ab\_ana} means the pairing \([\{a,f^2(c)\},\{b,f^2(d)\}]\).

On the other hand, the base program in Example 7 is represented as follows:

\[
\begin{array}{l}
\text{fact}(w_1,(p(a,a):-true)).
\text{fact}(w_1,(p(f(X),a):-p(X,a))).
\text{world}(w_1,w_2).
\end{array}
\]

Then, we obtain the following results:

\[
\begin{array}{c}
?\:- \text{ab\_ana}(p(f(f(c)),f(d)),X,P,W).
\text{no}
?\:- \text{ab\_ana}(p(f(f(c)),d),X,P,W).
\text{no}
?\:- \text{ab\_ana}(p(f(f(c)),c),X,P,W).
X = p(c,c),
P = [a\leftarrow c],
W = w_2;
\text{no}
?\:- \text{ab\_ana}(p(f(f(c)),f(c)),X,P,W).
X = p(f(c),f(c)),
P = [a\leftarrow f(c)],
W = w_2;
\text{no}
\end{array}
\]

Since we regard an analogy as a function, for the above two goals, we obtain no analogies. Then, we also obtain no deducible hypotheses.

\section{Conclusion}

In this paper, we have discussed rule-finding abduction for logic programming. For the termination of rule-finding abduction, we have defined two concepts of loop-pair and loop-elimination, and investigated the properties of them. Furthermore, we have discussed the relationship between analogical reasoning and rule-finding abduction. Then, we have formulated rule-finding abduction with analogy. We have shown that a deducible hypothesis is correct in the sense of analogical reasoning. Also we have shown that, if a target program is empty, then a deducible hypothesis is polynomial time computable on the length of a surprising fact and the size of a proof tree.

We have left some future works. For rule-finding abduction, we assume that the set of programs is given in advance. Then, we should consider how the set of programs is given. In particular, it is a future work how the set of programs is appropriate for rule-finding abduction. Furthermore, we should also show how the base and target programs are appropriate for rule-finding abduction with analogy.

This paper has also discussed rule-finding abduction and analogical reasoning in the same framework. This is a certain step toward acquiring the knowledge from abductive and analogical viewpoints, although we have just shown a few theoretical results. We need to formulate so called \textit{analogy by abduction} other than \textit{abduction with analogy}. We also need to solve the problem of incorporating rule-finding abduction with rule-generating abduction by using analogy.
References


