Rule-Generating Abduction for Recursive Prolog

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

出版情報：RIFIS Technical Report. 85, 1994-04. Research Institute of Fundamental Information Science, Kyushu University
バージョン：
権利関係：

Rule-Generating Abduction for Recursive Prolog

Kouichi Hirata
Research Institute of Fundamental Information Science
Kyushu University 33, Fukuoka 812, Japan
phone: +81-92-641-1101 ext. 4478 fax: +81-92-611-2668
e-mail: hirata@rifis.kyushu-u.ac.jp

Abstract

The rule-generating abduction is a kind of abduction which generates a rule and proposes a hypothesis from *a surprising fact*. In general, there may exist infinitely many rules and hypotheses to explain such a surprising fact. Hence, we need to put some restriction on the class of rules. In rule-generating abduction, only one surprising fact is given. Hence, we also need to generalize the concept of a surprising fact. When we deal with such generalizations, we must avoid overgeneralization. It should be determined whether or not a generalization is overgeneral by an intended model. However, it is hard to give in advance such an intended model in our rule-generating abduction. Hence, in this paper we introduce a syntactical formulation of generalization, in which it can be determined whether or not a generalization is overgeneral by the forms of atoms and substitutions. On the other hand, by the restriction of rules, it suffices to consider only two types of terms, constants and lists, and two types of substitutions with these two terms. By using the above generalizations and substitutions, we design an algorithm for rule-generating abduction, which generates rules and proposes hypotheses in polynomial time with respect to the length of a surprising fact. The number of rules and hypotheses is at most the number of common terms in a surprising fact. Furthermore, we show that a common term in some argument of a surprising fact also appears in the same argument of the proposed hypothesis by this algorithm.

1 Introduction

C. S. Peirce, who was a philosopher, scientist and logician, asserted that a scientific research consists of three stages, *abduction*, *deduction*, and *induction* [Pei65, Ino92, Yon82]. According to him, abduction is an inference which begins with an observation of *a surprising fact*, and proposes a hypothesis to explain why the fact arises. The main role of abduction is to propose a hypothesis. Thus, abduction is *a method of scientific discovery*. An inference schema by abduction is described by the following three steps [Pei65, Ino92, 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:

$$\frac{C \quad A \rightarrow C}{A}.$$

In computer science, especially in computational logic and logic programming, many researchers have extensively studied the abduction from various viewpoints. Plotkin [Plo71] has studied abduction together with inductive generalization. It is considered that Shapiro's model inference system [Sha81] and inductive logic programming [Lin89a, Lin89b, Mug92a, Mug92b] are the extensions of Plotkin. These are also related to machine learning and knowledge acquisition. On the other hand, Pople [Ino92, Kun87] has given one direction for the researches of abduction. It is considered that Poole's Theorist [Poo88, Ino92, Kun87], Kunifuji's hypothesis-based reasoning [Kun87], and abductive logic programming [Dun91, EK89, KM90] are the extensions of Pople. These are also related to knowledge representation.

In order to systematically understand these various researches of abduction, we have classified abduction into five types by an interpretation of syllogism [Hir93]; *rule-selecting abduction*, *rule-finding abduction*, *rule-generating abduction*, *theory-selecting abduction*, and *theory-generating abduction*. By this classification, the above researches are placed in the following positions [Hir93]: Abductive logic programming [Dun91, EK89, KM90] is a sort of rule-selecting abduction. The constructive operators such as V and W operators [Mug92b] in inductive logic programming are a sort of rule-generating abduction. Poole's Theorist [Poo88] and hypothesis-based reasoning [Kun87] are a sort of theory-selecting abduction. Shapiro's model inference system [Sha81] and inductive logic programming [Lin89a, Lin89b, Mug92a, Mug92b] are a sort of theory-generating abduction.

In this paper, we investigate the *rule-generating abduction*, which generates a rule and proposes a hypothesis from a surprising fact. In rule-generating abduction, only one surprising fact is given. Hence, we need to generalize the concept of a surprising fact.

A *generalization* is an important tool for inductive logic programming and program synthesis. Plotkin has introduced and developed the *least generalization* and the *relative least generalization* [Plo70, Plo71]. Arimura *et al.* have developed Plotkin's least generalization as the *minimal multiple generalization* [ASO91]. Note that all of them are researches on the generalization of at least two atoms. Thus, the following problem arises; Is the generalization of one atom worth or worthless? Hirowatari and Arikawa [HA94] have introduced the *partially isomorphic generalization* and answered this problem affirmatively in the framework of analogical reasoning.

When we deal with generalizations, we must avoid overgeneralization. It should be determined whether or not a generalization is overgeneral by an intended model. However, it is hard to give in advance such an intended model in our rule-generating abduction. Hence, in this paper we introduce a syntactical formulation of generalization, in which it can be determined whether or not a generalization is overgeneral by the forms of atoms and substitutions. In this formulation, a common ground term is replaced by a common variable, because an atom represents a relation which holds between its arguments.

This paper is organized as follows: In general, there may exist infinitely many rules and hypotheses to explain a surprising fact in rule-generating abduction. Hence, we need to put some restriction on the class of logic programs. In Section 2, we introduce the syntactical characterization of the rule $p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)$. Throughout this paper, we deal with these classes.

In Section 3, we formulate a *safe generalization*, in which it can be determined whether or not a generalization is overgeneral by the forms of atoms and substitutions, instead of an intended model. On the other hand, by the restriction of logic programs, it suffices to consider only two types of terms, constants and lists, and two types of substitutions with these two terms, *constant substitutions* and *list substitutions*. A constant substitution θ_c consists of the bindings $x := c$, where c is a constant symbol, while a list substitution θ_l consists of the bindings $x := l$, where l is a list. For these substitutions, we give a condition that the

generalization is safe with respect to the composition $\theta_c\theta_l$ of θ_c and θ_l.

In Section 4, by using the above generalizations and substitutions, we design an algorithm for rule-generating abduction, which generates rules and proposes hypotheses in polynomial time with respect to the length of a surprising fact. The number of rules and hypotheses is at most the number of common terms in a surprising fact. Furthermore, we show that a common term in some argument of a surprising fact also appears in the same argument of the proposed hypothesis by this algorithm.

In Section 5, we discuss the several examples for this algorithm.

2 Preliminary

Throughout this paper, we deal with the following class of programs:

$$P = \{p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)\}.$$

Hirata [Hir93] has introduced a class such that, for a given definite program P in the class and a ground atom α, all the derivations of $P \cup \{\leftarrow \alpha\}$ are finite. In this section, we reform the definitions [Hir93] for the above rule. In this paper, we assume that readers are familiar with the notions of logic programming and definite clause [Llo87]. We also assume that any term in $\mathcal{L}$ is either a constant or a list, where $\mathcal{L}$ is a first-order language with an n-ary predicate symbol p, a list constructor $[\_|\_]$, and finitely many constant symbols $[\,], a_1, \cdots, a_n$.

For a term t, $|t|$ denotes the length of t, that is, the number of all occurrences of symbols in t. In particular, for a list l, the length $|l|$ of l is defined as follows: $|l| = 1$, if l is an empty list $[\,]$. Otherwise $|l| = n + 1$, if t is a list $[a|list]$ and $|list| = n$.

In order to discuss the termination of derivations, we introduce some classes in the following way:

Definition 1 (Hirata [Hir93], Yamamoto [Yam92]) Let C be a clause $p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)$.

1. C is *head-reducing* if there exists an i such that $|t_i\theta| > |s_i\theta|$ for any ground substitution θ.
2. C is *weakly reducing* if $|t_i\theta| \geq |s_i\theta|$ for any ground substitution θ and for any i.
3. C is *weakly head-reducing* if it is head-reducing and weakly reducing.

In other words, a clause $p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)$ is weakly head-reducing if $|t_i\theta| \geq |s_i\theta|$ for any i, and $|t_k\theta| > |s_k\theta|$ for at least one argument k and for any ground substitution θ.

There are many Prolog programs for list processing such that any argument of the head is either x or $[w|x]$. Then, we restrict the form of clause as follows: A clause $p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)$ is 2-*reducing*, if t_i is either x_i or $[w_i|x_i]$ for any i, and it is head-reducing. A clause $p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)$ is *weakly* 2-*reducing* if it is weakly reducing and 2-reducing.

Example 1 The following Prolog programs in Sterling and Shapiro [SS86] are weakly 2-reducing.

```
list([W|X]):- list(X)
member(X,[W|Y]):- member(X,Y)
prefix([W|X],[W|Y]):- prefix(X,Y)
suffix(X,[W|Y]):- suffix(X,Y)
append([W|X],Y,[W|Z]):- append(X,Y,Z)
concat(X,[W|Y],[W|Z]):- concat(X,Y,Z)
```

Note that the clauses of `member` and `suffix` are the same forms. The first argument of `member` is a constant, while that of `suffix` is a list.

Then, the following theorem holds.

Theorem 1 (Hirata [Hir93], Yamamoto [Yam92]) Let α be a ground atom with a predicate p and C be a clause $p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)$.

1. If C is head-reducing, in particular 2-reducing, then the derivation of $\{p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)\} \cup \{\leftarrow \alpha\}$ is finite.
2. If C is weakly head-reducing, in particular weakly 2-reducing, then the derivation of $\{p(t_1, \cdots, t_n) \leftarrow p(s_1, \cdots, s_n)\} \cup \{\leftarrow \alpha\}$ is finite, and all nodes of derivation are ground goals.

3 Safe Generalization

It is an important point to avoid overgeneralization when we deal with generalization. In general, it is determined whether or not a generalization β of α is overgeneral is determined by an intended model. Let $\beta\theta = \alpha$ and M be an intended model. Then, β is an overgeneralization of α if there exists a ground atom γ such that $\forall\beta \vdash \gamma$ and $M \not\models \gamma$ for an intended model M. However, the decision problem of whether or not there exists such a γ is undecidable. On the other hand, in rule-generating abduction, only one surprising fact is given, and it is hard to give in advance an intended model. To overcome these difficulties, we introduce a syntactical formulation of generalization.

Let θ be a ground substitution, that is, $\theta = \cup_{i=1}^{n}\{x_i := t_i\}$, where any t_i is a ground term. Let α be a ground atom and β be an atom such that $\beta\theta = \alpha$. Note that, throughout this paper, if $\beta\theta = \alpha$ then a variable x_i appears in β and $t_i \neq [\,]$. A substitution θ is *well-defined* if, for any t_i, there exists no term t_j such that t_j is a subterm of t_i. In general, β is called an *overgeneralization* of α if θ is not well-defined.

Example 2 Let α be a ground atom $p([a,b],[b])$. Let

$$\beta_1 = p([a,x],[b]),\ \beta_2 = p([a|x],[b]),\ \beta_3 = p([x|y],y),$$
$$\beta_4 = p([a|x],[y]),\ \beta_5 = p([a|x],y).$$

For any β_i, there exist the following substitutions θ_i such that $\beta_i\theta_i = \alpha$:

$$\theta_1 = \{x := b\},\ \theta_2 = \{x := [b]\},\ \theta_3 = \{x := a, y := [b]\},$$
$$\theta_4 = \{x := [b], y := b\},\ \theta_5 = \{x := [b], y := [b]\}.$$

Then, θ_1, θ_2, and θ_3 are well-defined, while θ_3 and θ_4 are not.

Let α be a ground atom and β be an atom such that $\beta\theta = \alpha$. If a substitution $\theta = \cup_{i=1}^{n}\{x_i := t_i\}$ is well-defined, then we can define the reversal $\theta^{-1} = \cup_{i=1}^{n}\{t_i := x_i\}$. Note that, if θ is well-defined, then, for any t_i and x_i, there exists no term t_j such that t_j is a subterm of t_i and no variables x_i such that $x_i = x_j (j \neq i)$. However, even if θ is well-defined, β is not always $\alpha\theta^{-1}$.

Example 3 For β_1 and β_2 in Example 2,

$$\alpha\theta_1^{-1} = p([a,b],[b])\{b := x\} = p([a,x],[x]) \neq \beta_1,$$
$$\alpha\theta_2^{-1} = p([a,b],[b])\{[b] := x\} = p([a|x],x) \neq \beta_2.$$

On the other hand, $\alpha\theta_3^{-1} = p([a,b],[b])\{a := x, [b] := y\} = p([x|y], y) = \beta_3$.

For the reversal θ^{-1}, the following lemma holds.

Lemma 1 Let α be a ground atom and β be an atom such that $\beta\theta = \alpha$. Suppose that a substitution $\theta = \cup_{i=1}^{n}\{x_i := t_i\}$ is well-defined. Then, $\beta = \alpha\theta^{-1}$ if and only if no t_i appears in β.

Proof Suppose $\beta = \alpha\theta^{-1}$. By the definition of θ^{-1}, $\alpha\theta^{-1}$ is an atom which replaces all t_i in α with variable x_i. Then, no t_i appears in $\alpha\theta^{-1} = \beta$.

For simplicity, suppose that $\theta = \{x := t\}$. If t appears once in α, that is $\alpha = p(\cdots t \cdots)$, then $\alpha\theta^{-1} = p(\cdots t \cdots)\{t := x\} = p(\cdots x \cdots)$. Since α is ground, $\alpha\theta^{-1} = \beta$.

If t appears at least twice in α, that is $\alpha = p(\cdots t \cdots t \cdots)$, then β is $p(\cdots x \cdots y \cdots)$. If $x \neq y$, then, by $\beta\theta = \alpha$, θ is $\{x := t, y := t\}$. Since θ is not well-defined, it is contradiction. Then, $x = y$, $\theta = \{x := t\}$, and $\beta = p(\cdots x \cdots x \cdots)$. Hence, $\beta = \alpha\theta^{-1}$. □

A ground term $t(\neq [\,])$ is a *common term* in α if t appears at least twice in α except an empty list $[\,]$. In particular, if a common term is a ground list, it is called a *common list*.

Definition 2 Let α be a ground atom, θ be a substitution $\cup_{i=1}^{n}\{x_i := t_i\}$, and β be an atom such that $\beta\theta = \alpha$. An atom β is a *safe generalization* of α if (β, θ) satisfies the following *safeness conditions*:

1. θ is well-defined,
2. $\beta = \alpha\theta^{-1}$, and
3. if there exist common terms in α, then there exists a ground term $t_j \in \cup_{i=1}^{n}\{t_i\}$ such that t_j is a common term in α.

The safeness condition 1 means that β is not overgeneral on α. The safeness condition 2 and 3 mean that a generalization of β is not overgeneral on α. Hence, if a given ground atom is generalized *safely*, then we can avoid overgeneralization. Furthermore, each of safe generalizations is corresponding to the relation which is represented by a given ground atom.

Let α be a ground atom and β be an atom such that $\beta\theta = \alpha$. Let t be some common term in α. If θ is well-defined and θ^{-1} is the form $\{t := x\}$, then β is safe on α.

Example 4 Let α be a ground atom $p([a,b],[b])$ and β be an atom such that $\beta\theta = \alpha$. Then, the common terms in α are $[b]$ and b.

1. Let β_1 be an atom $p(x,y)$ and θ_1 be a substitution $\theta_1 = \{x := [a,b], y := [b]\}$. By the safeness condition 1, β_1 is not safe on α.
2. Let β_2 be an atom $p([x,b],[y])$ and θ_2 be a substitution $\{x := a, y := b\}$. By the safeness condition 2, β_2 is not safe on α.
3. Let β_3 be an atom $p(x,[b])$ and θ_3 be a substitution $\{x := [a,b]\}$. By the safeness condition 3, β_3 is not safe on α.

For the above α, atoms $p([a|x],x), p([a,x],[x]), p([y,x],[x])$, and $p([y|x],x)$ are safe on α.

In general, an atom represents a relation which holds between its arguments. Thus, the syntactical generalization of one atom should obtained by replacing a common term by a common variable. The above safe generalization is an example of such generalizations. On the other hand, it suffices to consider only two types of terms, constants and lists. Then, we also define the following two types of substitution.

Let θ be a substitution $\cup_{i=1}^{n}\{x_i := t_i\}$. Then, θ is a *constant substitution* (*resp.* a *list substitution*) if every t_i is a constant symbol (*resp.* a ground list) without an empty list [ ]. In particular, a constant substitution is related to *partially isomorphic generalizations* which have been introduced by Hirowatari and Arikawa [HA94].

Let α be an atom. A term t is a *replaceable term* of α if t is a constant symbol. For a replaceable term t of α, let $\alpha[t]$ be an atom obtained by replacing each t in α by a new variable Z which does not appear in α. Then, we write $\alpha \to \beta$ when $\alpha[t]$ is a variant of β. We define $\to^*$ as the reflexive and transitive closure of $\to$.

Definition 3 (Hirowatari and Arikawa [HA94]) Let α and β be atoms. Then, β is a *partially isomorphic generalization* of α if $\alpha \to^* \beta$.

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 $\beta \in S$, α is a variant of β.

Theorem 2 (Hirowatari and Arikawa [HA94]) Let α be an atom and S be the set of all partially isomorphic generalizations of α. Then, $[S]$ is a lattice whose partial order is $\to^*$, meet operator is the greatest instantiation, and join operator is the least generalization.

Note that, though a replaceable term includes an empty list [ ], the definition of a replaceable term is independent of the proof of Theorem 2. Let RT be the set of all replaceable terms of α and $T \subseteq RT$. Then, we can re-formulate a partially isomorphic generalization by using T instead of RT, and the above Theorem 2 also holds for a set T of replaceable terms. Let α be a ground atom, θ_c be a constant substitution, and β be an atom such that $\beta\theta_c = \alpha$. Thus, we assume that *β is a partially isomorphic generalization of α, whose replaceable terms are all constant symbols in α except an empty list* [ ].

In the next section, we apply rule-generating abduction to a list substitution θ_l and a constant substitution θ_c in the following way: Let α be a ground atom, i.e., a surprising fact. First, by using a list substitution, we obtain an atom β such that $\beta\theta_l = \alpha$ and β is safe on α. Secondly, by using a constant substitution, we obtain an atom γ such that $\gamma\theta_c = \beta$ and γ is safe on β. By the above assumption, γ is also a partially isomorphic generalization of β.

Unfortunately, $\theta_c\theta_l$ is not always well-defined, and γ is not always safe on α. For example, let α be a ground atom $p([a,b,c],[b,c],[a,b,c])$. Then, there exist the following atoms β_i such that $\beta_i\theta_i = \alpha$ and θ_i is a well-defined list substitution:

$$\begin{array}{ll} \beta_1 = p([a,b|x],[b|x],[a,b|x]) & \theta_1 = \{x := [c]\}, \\ \beta_2 = p([a|y],y,[a|y]) & \theta_2 = \{y := [b,c]\}, \\ \beta_3 = p(z,[b,c],z) & \theta_3 = \{z := [a,b,c]\}. \end{array}$$

There exists no atom γ and substitutions $\sigma(\neq \varepsilon)$ such that $\gamma\sigma = \beta_3$ and $\theta_3\sigma$ is well-defined. Note that there does not exist the greatest list generalization of α.

Let α be a ground atom. Let β and γ be atoms such that $\beta\theta_l = \alpha$ and $\gamma\theta_c = \beta$. Suppose that both (β, θ_l) and (γ, θ_c) satisfy the safeness conditions. Then, the following two theorems hold.

Theorem 3 If $\theta_c\theta_l$ is well-defined, then γ is a safe generalization of α.

Proof Suppose that $\theta_c\theta_l$ is well-defined. Then, $(\gamma, \theta_c\theta_l)$ satisfies the safeness condition 1.

Since both (β, θ_l) and (γ, θ_c) satisfy the safeness conditions, $(\gamma, \theta_c\theta_l)$ satisfies the safeness condition 3.

By supposition, $\beta = \alpha\theta_l^{-1}$ and $\gamma = \beta\theta_c^{-1}$. The list substitution θ_l replaces the common lists in α by variables. The constant substitution θ_c replaces the same constant symbols in β by the same variables and other constant symbols by other variables. Hence, the composition $\theta_c\theta_l$ replaces the common lists in α by variables, the same constant symbols in α except common terms by the same variables, and other constant symbols by other variables. By Lemma 1 and well-definedness of $\theta_c\theta_l$, $(\gamma, \theta_c\theta_l)$ satisfies the safeness condition 2. □

Theorem 4 Suppose that any constant which appears in common lists in α does not appear in α except them. If $\beta = \alpha\theta_l^{-1}$ and $\gamma = \beta\theta_c^{-1}$, then $\theta_c\theta_l$ is well-defined. Hence, γ is a safe generalization of α.

Proof Suppose that $\theta_l = \cup_{i=1}^{n}\{x_i := l_i\}$, where l_i is a common list in α. For any j-th argument's term t_j of α, if t_j includes l_i, then $t_j = [a_1^j, a_2^j, \cdots, a_{n_j}^j | l_i]$, and no constants $a_1^j, a_2^j, \cdots, a_{n_j}^j$ appear in l_i. Then, θ_c does not include the binding $x := c$ such that c appears in l_i. Hence, $\theta_c\theta_l$ is well-defined. By Theorem 3, $(\gamma, \theta_c\theta_l)$ satisfies the safeness conditions. Then, for γ such that $\gamma\theta_c\theta_l = \alpha$, γ is a safe generalization of α. □

4 Rule-Generating Abduction

The *rule-generating abduction* is a process which generates a rule and proposes a hypothesis from a surprising fact. An inference schema by rule-generating abduction is described by the following three steps:

1. A ground atom C is given.
2. A rule $C' \leftarrow A'$ is generated, where $C'\theta = C$ and $A'\theta = A$.
3. A hypothesis A is proposed.

In Section 2, we have discussed the head-reducing clause of which all the derivations are finite. Suppose that the rule $p(s'_1, \cdots, s'_n) \leftarrow p(t'_1, \cdots, t'_n)$ is head-reducing. Then, for a given ground atom $p(t_1, \cdots, t_n)$, the head-reducing rule

$$p(t'_1, \cdots, t'_n) \leftarrow p(s'_1, \cdots, s'_n)$$

is generated and the hypothesis $p(s_1, \cdots, s_n)$ is proposed by rule-generating abduction, where $p(t'_1, \cdots, t'_n)\theta = p(t_1, \cdots, t_n)$ and $p(s'_1, \cdots, s'_n)\theta = p(s_1, \cdots, s_n)$. An inference schema is depicted by the following syllogism:

$$\frac{p(t_1, \cdots, t_n) \quad p(t'_1, \cdots, t'_n) \leftarrow p(s'_1, \cdots, s'_n)}{p(s_1, \cdots, s_n)}.$$

Unfortunately, the number of head-reducing rules is at most

$$(m-1)^n \sum_{i=1}^{n} \frac{n!}{i!},$$

where $m = \max_{1 \le i \le n} |t_i|$ for a ground atom $p(t_1, \cdots, t_n)$.

In usual, there exists no variable which appears only once in the body of the rule $p(t'_1, \cdots, t'_n) \leftarrow p(s'_1, \cdots, s'_n)$. Then, the number of weakly head-reducing rules is at most m^n, where $m = \max_{1 \le i \le n} |t_i|$ for a ground atom $p(t_1, \cdots, t_n)$.

Furthermore, suppose that a given program is 2-reducing or weakly 2-reducing. Then, for a ground atom with n-ary predicate symbol, the number of 2-reducing rules is at most

$$\sum_{i=1}^{n} \frac{n!}{i!},$$

while the number of weakly 2-reducing rules is at most 2^n.

On the other hand, by using safe generalizations in Section 3, we design an algorithm to generate weakly 2-reducing rules as follows: Suppose that a ground atom α is given. First, by generalizing α with a list substitution θ_l, we obtain an atom β such that $\beta\theta_l = \alpha$ and β is safe on α. We call such β a *list generalization* of α. Secondly, by generalizing β with a constant substitution θ_c, we obtain an atom γ such that $\gamma\theta_c = \beta$ and γ is safe on β. We call such γ a *constant generalization* of β. Note that γ is also assumed the *partially isomorphic generalization* of β. For this algorithm, the number of rules is at most the number of generalizations. Then, we investigate the number of generalizations.

Let α be a ground atom $p(t_1, \cdots, t_n)$. For all common lists in α, we can classify them by the *sublist relation*. For example, let α be a ground atom $p([a,b,c,d],[c,d],[b,c],[c],[b,c,d])$ and t_i be the i-th argument's term of α. Then, t_2, t_4, and t_5 are common lists in α. By the sublist relation, we classify them into $\{t_2, t_5\}$ and $\{t_4\}$. Let l be the number of such classes. Then, the number of the maximal generalizations is at most

$$\left(\frac{\left(\sqrt{2}\right)^n}{l} \right)^l.$$

Even if $l = 1$, the number of the maximal list generalizations of α is at most $\left(\sqrt{2}\right)^n$. We discuss more detail in Appendix at the end of this paper.

The number of the maximal list generalizations increases exponentially with respect to n. Thus, in the following algorithm *PROPOSE*, we restrict the reversal of list generalizations to the form $\{t := x\}$, where t is both a common term in α and some argument's term of α. Obviously, the generalization $\alpha\{x := t\}$ is safe on α. The number of weakly 2-reducing rules generated by the algorithm *PROPOSE* is at most n.

An important basis on the algorithm *PROPOSE* is that, *if the i-th argument's term is some common list in α, then the i-th argument's term of the head of the generated rule is a variable; otherwise, it is a list.* Furthermore, by the algorithm *PROPOSE*, the rules in Example 1 are constructed from one ground atom.

In *PROPOSE*, $rs\_abd(fact, head \leftarrow body, hyp)$, which is rule-selecting abduction, is a procedure to propose a hypothesis hyp such that $(head)\sigma = fact$ and $(body)\sigma = hyp$ for some substitution σ. Then, the following two theorems hold.

Theorem 5 Let α be a ground atom $p(t_1, \cdots, t_n)$ and $k = |t_1| + \cdots + |t_n|$. Then, the algorithm *PROPOSE* computes the rules and hypotheses in $O(k^3)$ time.

Proof For any t_i, it can be determined whether or not t_i is a sublist of t_j in $O(|t_i|)$. Then, for any i, it can be determined whether or not t_i is a sublist of any $t_j (j \neq i)$ in $O((n-1)|t_i|)$. Hence, the set L in the algorithm *PROPOSE* can be constructed in $O((n-1)k)$.

For the selected β in L, the greatest constant generalization of β is also a partially isomorphic generalization of β. By Hirowatari and Arikawa [HA94], a partially isomorphic generalization γ of β can be computed in $O(k^2)$. Since the procedures in the for-loop can be computed in $O(n)$, the for-loop terminates in $O(n^2)$. Then, the procedures in while-loop can be computed in $O(k^2 + n^2)$. Since the number of elements in L is at most n, the while-loop terminates in $O(k^2 n + n^3)$. Hence, the algorithm *PROPOSE* terminates in $O((n-1)k + k^2 n + n^3)$.

Since $n \leq k$, the algorithm *PROPOSE* computes rules and hypotheses in $O(k^3)$ time. □

Algorithm *PROPOSE*
input $\alpha = p(t_1, \cdots, t_n)$: fact, i.e., ground atom
output $head \leftarrow body$: rule
δ : hypothesis
$L := \{\beta \mid \beta = \alpha\{t_i := v_i\}, t_i$: common list in $\alpha\} \cup \{\alpha\}$;
/* β : safe on α */
while $L \neq \phi$ **do**
select $\beta \in L$;
$\gamma = p(s_1, \cdots, s_n)$: the greatest constant generalization of β;
for $i = 1$ **to** n
if $s_i = [\,]$ **then** /* base step */
output $\gamma \leftarrow true$: rule
output $true$: hypothesis
halt;
else if s_i: a variable **then** /* induction step */
$head\_arg_i := x_i$; /* x_i is a new variable */
else /* $s_i = [w_1^i, \cdots]$ */
$head\_arg_i := [w_1^i | x_i]$; /* x_i is a new variable */
end
end
$head := p(head\_arg_1, \cdots, head\_arg_n)$;
/* $head\_arg_i = [w_1^i | x_i]$ or x_i */
$body := p(x_1, \cdots, x_n)$;
output $head \leftarrow body$: rule
$rs\_abd(\alpha, head \leftarrow body, \delta)$;
output δ : hypothesis
$L := L - \{\beta\}$
end

Theorem 6 Let α be a ground atom $p(t_1, \cdots, t_n)$. If there exists a common list l in α and l appears in t_i, then l also appears in the i-th argument of the proposed hypothesis δ by the algorithm *PROPOSE*.

Proof Let l be a common list in α. If some argument's term t_i of α is l, then each of the i-th argument's terms of the head and of the body is a variable x_i by the algorithm *PROPOSE*. Then, the i-th argument's term of δ is also l.

If l appears in some argument's term t_i of α, then $t_i = [a_1^i, a_2^i, \cdots, a_{n_i}^i | l]$. By the algorithm *PROPOSE*, the i-th argument's term of the head is a list $[y_1^i | x_i]$, where y_1^i is a variable corresponding to a_1^i, while one of the body is a variable x_i. Then, for the hypothesis δ, the i-th argument's term of δ is $[a_2^i, \cdots, a_{n_i}^i | l]$.

Hence, a common list also appears in the same argument of the proposed hypothesis δ by the algorithm *PROPOSE*. □

Hence, if a given ground atom satisfies the relation on common lists, then the proposed hypothesis by the algorithm *PROPOSE* also satisfies it.

5 Examples

In this section, we discuss the several examples for the algorithm *PROPOSE*.

Example 5 Let α be a ground atom $p([a,b],[c,d],[a,b,c,d])$. The list $[c,d]$ is both a common list in α and the second argument's term of α. By the construction of L, $\beta = p([a,b],v_2,[a,b|v_2])$ is a *safe* list generalization of α, and $\gamma = p([x,y],v_2,[x,y|v_2])$ is the greatest constant generalization of β. The first argument's term of γ is a list which begins with x, the second argument's term is a variable v_2, and the third argument's term is also a list which begins with x. By the for-loop in *PROPOSE*, we obtain the head $p([x|x_1],x_2,[x|x_3])$ and the body $p(x_1,x_2,x_3)$. Hence, *PROPOSE* generates the rule

$$p([x|x_1],x_2,[x|x_3]) \leftarrow p(x_1,x_2,x_3)$$

and proposes the hypothesis $p([b],[c,d],[b,c,d])$. Note that the predicate p means `append` in Example 1.

Since L includes α, let $\beta = \alpha$. Then, $\gamma = p([x,y],[z,w],[x,y,z,w])$. The first argument's term of γ is a list which begins with x, the second argument's term is a list which begins with z, and the third argument's term is also a list which begins with x. By the for-loop in *PROPOSE*, we obtain the head $p([x|x_1],[z|x_2],[x|x_3])$ and the body $p(x_1,x_2,x_3)$. Hence, *PROPOSE* generates the rule

$$p([x|x_1],[z|x_2],[x|x_3]) \leftarrow p(x_1,x_2,x_3)$$

and proposes the hypothesis $p([b],[d],[b,c,d])$.

Furthermore, each of the rules and hypotheses by *PROPOSE* satisfies the following syllogism respectively:

$$\frac{p([a,b],[c,d],[a,b,c,d])\quad p([x|x_1],x_2,[x|x_3]) \leftarrow p(x_1,x_2,x_3)}{p([b],[c,d],[b,c,d])},$$

$$\frac{p([a,b],[c,d],[a,b,c,d])\quad p([x|x_1],[z|x_2],[x|x_3]) \leftarrow p(x_1,x_2,x_3)}{p([b],[d],[b,c,d])}.$$

Example 6 Let α be a ground atom $p(a,[a,b])$. Since there exists no common list in α, $\beta = p(a,[a,b])$ and $\gamma = p(x,[x,y])$. The first argument's term of γ is a variable x, and the second argument's term is a list which begins with x. By the for-loop in *PROPOSE*, we obtain the head $p(x_1,[x|x_2])$ and the body $p(x_1,x_2)$. Hence, *PROPOSE* generates the rule

$$p(x_1,[x|x_2]) \leftarrow p(x_1,x_2)$$

and proposes the hypothesis $p(a,[b])$. Note that the predicate p means `member` in Example 1.

Let α be a ground atom $p([a],[a,b])$. Since there exists no common list in α, $\beta = p([a],[a,b])$ and $\gamma = p([x],[x,y])$. The first argument's term of γ is a list which begins with x, and the second argument's term is also a list which begins with x. By the for-loop in *PROPOSE*, we obtain the head $p([x|x_1],[x|x_2])$ and the body $p(x_1,x_2)$. Hence, *PROPOSE* generates the rule

$$p([x|x_1],[x|x_2]) \leftarrow p(x_1,x_2)$$

and proposes the hypothesis $p([\,],[b])$. Note that the predicate p means `prefix` in Example 1.

Let α be a ground atom $p([b],[a,b])$. The list $[b]$ is both a common list in α and the first argument's term of α. Then, $\beta = p(v_1,[a|v_1])$ and $\gamma = p(v_1,[x|v_1])$. The first argument's term of γ is a variable v_1, and the second argument's term is a list which begins with x. By the for-loop in *PROPOSE*, we obtain the head $p(x_1,[x|x_2])$ and the body $p(x_1,x_2)$. Hence, *PROPOSE* generates the rule

$$p(x_1,[x|x_2]) \leftarrow p(x_1,x_2)$$

and proposes the hypothesis $p([b],[b])$. Note that the predicate p means `suffix` in Example 1. On the other hand, since L includes $\alpha = p([a,b],[b])$, let $\beta = \alpha$. Then, $\gamma = p([x],[y,x])$. The first argument's term of γ is a list which begins with x, and the second argument's term is a list which begins with y. By the for-loop in *PROPOSE*, we obtain the head $p([x|x_1],[y|x_2])$ and the body $p(x_1,x_2)$. Hence, *PROPOSE* generates the rule

$$p([x|x_1],[y|x_2]) \leftarrow p(x_1,x_2)$$

and proposes the hypothesis $p([\,],[b])$.

If $\alpha = p([a,b],[c,d],[a,b,c,d])$, then *PROPOSE* generates the rule

$$p([x|x_1],x_2,[x|x_3]) \leftarrow p(x_1,x_2,x_3)$$

and proposes the hypothesis $p([b],[c,d],[b,c,d])$. If $\alpha = p([a,b],[a,b,c,d],[c,d])$, then *PROPOSE* also generates the rule

$$p([x|x_1],[x|x_2],x_3) \leftarrow p(x_1,x_2,x_3)$$

and proposes the hypothesis $p([b],[b,c,d],[c,d])$. Hence, the algorithm *PROPOSE* is independent of the order of argument. Furthermore, as `member` and `suffix` in Example 6, the algorithm *PROPOSE* is also independent of the types of argument. In other words, *PROPOSE* needs no types of argument.

We have implemented the algorithm *PROPOSE* by Prolog. The predicate `propose` returns the generated rule as the third argument. It also returns the hypothesis proposed by the generated rule as the second argument.

```
: ?- propose(p([a,b],[c,d],[a,b,c,d]),X,Y).
X = p([b],[d],[b,c,d]),
Y = p([_2046|_2602],[_1378|_2600],[_2046|_2598]):-p(_2602,_2600,_2598) ;
X = p([b],[c,d],[b,c,d]),
Y = p([_1578|_2070],_2058,[_1578|_2066]):-p(_2070,_2058,_2066) ;
no
: ?- propose(p(a,[a,b]),X,Y).
X = p(a,[b]),
Y = p(_1334,[_972|_1340]):-p(_1334,_1340) ;
no
: ?- propose(p([a],[a,b]),X,Y).
X = p([],[b]),
Y = p([_1016|_1418],[_1016|_1416]):-p(_1418,_1416) ;
no
: ?- propose(p([b],[a,b]),X,Y).
X = p([],[b]),
Y = p([_1158|_1560],[_904|_1558]):-p(_1560,_1558) ;
X = p([b],[b]),
Y = p(_1220,[_872|_1226]):-p(_1220,_1226) ;
no
: ?- propose(p([a,b,c],[b,c],[a,b,c]),X,Y).
X = p([b,c],[c],[b,c]),
Y = p([_2066|_2622],[_1738|_2620],[_2066|_2618]):-p(_2622,_2620,_2618) ;
X = p([b,c],[b,c],[b,c]),
Y = p([_1638|_2102],_2090,[_1638|_2098]):-p(_2102,_2090,_2098) ;
X = p([a,b,c],[c],[a,b,c]),
Y = p(_2360,[_1670|_2368],_2356):-p(_2360,_2368,_2356) ;
no
```

Note that, in the last example, $\beta_1 = p([a|v_2], v_2, [a|v_2])$ and $\beta_2 = p(v_1, [b,c], v_1)$ are the safe generalizations of $p([a,b,c],[b,c],[a,b,c])$. Hence, there are three hypotheses and rules for a ground atom $p([a,b,c],[b,c],[a,b,c])$.

On the other hand, for a ground atom $p([a,b,c],[d,e],[a,b,c])$, the predicate `propose` returns the following two rules and hypotheses:

```
: ?- propose(p([a,b,c],[d,e],[a,b,c]),X,Y).
X = p([b,c],[e],[b,c]),
Y = p([_2510|_3066],[_1490|_3064],[_2510|_3062]):-p(_3066,_3064,_3062) ;
X = p([a,b,c],[e],[a,b,c]),
Y = p(_2056,[_1366|_2064],_2052):-p(_2056,_2064,_2052) ;
no
```

6 Conclusion

In this paper, we have formulated an overgeneralization and a safe generalization, and given the algorithm *PROPOSE* to construct weakly 2-reducing rules. We have shown that the algorithm *PROPOSE* can generate rules and propose hypotheses in polynomial time with respect to the length of a surprising fact. Also we have shown that a common term in some argument of a surprising fact appears in the same argument of the proposed hypothesis by *PROPOSE*.

Abduction is an inference to propose hypotheses to be used before deduction and induction. We have left as a future work to combine abduction and inductive logic programming. Ling [Lin89a, Lin89b] has introduced the constructive inductive logic programming. There may exist a relationship between such works and this paper.

Furthermore, as to the inductive logic programming, we have left the problem of predicate invention. The number of rules and hypotheses becomes exponentially large. Hence, we need to introduce some heuristics such as on a distinction between a necessary and useful intermediate term, the number of local variables, invented predicate symbols and rules, and so on.

Acknowledgment

The author would like to thank Setsuo Arikawa and Eiju Hirowatari for many precious discussions about partially isomorphic generalizations. He also thanks the reviewers for valuable comments.

References

[ASO91] Arimura, H., Shinohara, T., Otsuki, S.: A polynomial time algorithm for finite unions of tree pattern languages. In Proceedings of the 2nd Workshop on Algorithmic Learning Theory (1991) 105–114.

[Dun91] Dung, P. M.: Negation as hypothesis: an abductive foundation for logic programming. In Proceedings of the 8th International Conference on Logic Programming (1991) 3–17.

[EK89] Eshghi, K., Kowalski, R. A.: Abduction compared with negation by failure. In Proceedings of the 6th International Conference on Logic Programming (1989) 234–254.

[Hir93] Hirata, K.: A classification of abduction: abduction of logic programming. Machine Intelligence **14** (to appear).

[HA94] Hirowatari, E., Arikawa, S.: Partially isomorphic generalization and analogical reasoning. In Proceedings of European Conference on Machine Learning (1994), Lecture Notes in Artificial Intelligence **784** (1994) 363–366.

[Ino92] Inoue, K.: Principles of abduction. Journal of Japanese Society for Artificial Intelligence **7** (1992) 48–59 (in Japanese).

[KM90] Kakas, A. C., Mancarella, P.: Generalized stable models: a semantics for abduction. In Proceedings of the 9th European Conference on Artificial Intelligence (1990) 385–391.

[Kun87] Kunifuji, S.: Hypothesis-based reasoning. Journal of Japanese Society for Artificial Intelligence **2** (1987) 22–87 (in Japanese).

[Lin89a] Ling, X.: Learning and invention of Horn clause theories - a constructive method. Methodologies for Intelligent Systems **4** (1989) 323–331.

[Lin89b] Ling, X.: Inventing theoretical terms in inductive learning of functions - search and constructive methods. Methodologies for Intelligent Systems **4** (1989) 332–341.

[Llo87] Lloyd, J. W.: Foundations of logic programming (second, extended edition). Springer-Verlag (1987).

[Mug92a] Muggleton, S. (ed.): Inductive logic programming. Academic Press (1992).

[Mug92b] Muggleton, S.: Machine invention of first-order predicates by inverting resolution. In Proceedings of the 5th International Conference on Machine Learning (1988) 339-352; In [Mug92a].

[Pei65] Peirce, C. S.: Collected papers of Charles Sanders Peirce (1839-1914). Hartshone, C. S., Weiss, P.(eds.), The Belknap Press (1965).

[Plo70] Plotkin, G. D.: A note on inductive generalization. Machine Intelligence **5** (1970) 153–163.

[Plo71] Plotkin, G. D.: A further note on inductive generalization. Machine Intelligence **6** (1971) 101–124.

[Poo88] Poole, D.: A logical framework for default reasoning. Artificial Intelligence **36** (1988) 27–47.

[Sha81] Shapiro, E. Y.: Inductive inference of theories from facts. Research Report **192**, Yale University (1981).

[SS86] Sterling, L., Shapiro, E.: The art of Prolog. The MIT Press (1986).

[Yam92] Yamamoto, A.: Procedural semantics and negative information of elementary formal system. Journal of Logic Programming **13** (1992) 89–97.

[Yon82] Yonemori, Y.: Peirce's semiotics. Keisou Syobou (1982) (in Japanese).

Appendix

In Section 4, we investigate the number of the maximal list generalizations. In this appendix, we explain how to solve the upper bound of this number.

Let α be a ground atom $p(t_1, \cdots, t_n)$ and K_n be the number of the maximal list generalizations of α. In Section 4, we introduce the classification by the sublist relation. Suppose that l is the number of such classes. If $l = 1$, then we can find the upper bound of K_n in the following way.

For simplicity, suppose that the common lists in α are $t_1, t_2, \cdots, t_{n-1}$, and $|t_1| > |t_2| > \cdots > |t_{n-1}|$. We denote the generalization $\alpha\{t_{j_1} := x_{j_1}, \cdots, t_{j_f} := x_{j_f}\}$ by $\beta_{(j_1,\cdots,j_f)}$. Note that t_{j_i+1} is not a common list in $\beta_{(j_i)}$. Furthermore, for $\beta_{(j_i,j_i+a,j_i+a+b)}$ $(a, b = 2 \text{ or } 3)$, there exist substitutions θ_{j_i+a+b}, θ_{j_i+a}, and θ_{j_i} such that

$$\begin{aligned}
\beta_{(j_i,j_i+a)} &= \beta_{(j_i,j_i+a,j_i+a+b)}\theta_{j_i+a+b}, \\
\beta_{(j_i,j_i+a+b)} &= \beta_{(j_i,j_i+a,j_i+a+b)}\theta_{j_i+a}, \\
\beta_{(j_i+a,j_i+a+b)} &= \beta_{(j_i,j_i+a,j_i+a+b)}\theta_{j_i}.
\end{aligned}$$

Hence, $\beta_{(j_i,j_i+a)}$, $\beta_{(j_i,j_i+a+b)}$, and $\beta_{(j_i+a,j_i+a+b)}$ are not maximal list generalizations.

By using the index of β, K_n is equal to the number of the sequences $(j_1,\cdots,j_f)$ which satisfy the following conditions:

1. $j_1 = 1$ or 2,
2. $j_f = n-1$ or n, and
3. the adjacent number of j_i is either j_i+2 or j_i+3.

For example, let $n = 8$. Then, the following seven sequences

$$(1,3,5,7),(1,3,6),(1,4,6),(1,4,7),(2,4,6),(2,4,7),(2,5,8)$$

satisfy the above conditions. For the sequence $(j_1,\cdots,j_f)$ which satisfies the above conditions, the number of sequences such that $j_1 = 1$ is greater that $j_1 = 2$. Let A_n be the set of the sequences $(j_1,\cdots,j_f)$ which satisfy the above conditions and $j_1 = 1$. Then, $K_n \leq 2|A_n|$. Furthermore, for $n \geq 6$, we can construct the set A_n in the following way:

1. if $(j_1,\cdots,j_f) \in A_{n-2}$, then $(j_1,\cdots,j_f,n) \in A_n$, and
2. if $(j'_1,\cdots,j'_f) \in A_{n-3}$, then $(j'_1,\cdots,j'_f,n-1) \in A_n$.

Hence, $|A_n|$ satisfies the following equations:

$$|A_3| = 1,\ |A_4| = 1,\ |A_5| = 2,\ |A_n| = |A_{n-2}| + |A_{n-3}|\ (n \geq 6).$$

By the mathematical induction, we obtain the following formula:

$$\left(\sqrt[3]{2}\right)^{n-2} \leq |A_n| \leq \left(\sqrt{2}\right)^{n-2}\ (n \geq 6).$$

Hence, the number K_n of the maximal generalizations is characterized by the following formula:

$$K_n \leq 2\left(\sqrt{2}\right)^{n-2} = \left(\sqrt{2}\right)^{n}.$$

Note that this formula holds for any $n \geq 1$.

Let l be the number of classes by the sublist relation and C_j be such class for $1 \leq j \leq l$. For any C_j, the number of the sequences which satisfy the above conditions is at most $\left(\sqrt{2}\right)^{|C_j|}$. Then, the number K_n of the maximal generalizations is at most $\left(\sqrt{2}\right)^{|C_1|} \times \cdots \times \left(\sqrt{2}\right)^{|C_l|}$. Hence, K_n is also characterized by the following formula:

$$K_n \leq \left(\frac{\left(\sqrt{2}\right)^{n}}{l}\right)^{l}.$$