プログラミング－言語・基礎・実践－研究報告　9－6
（1992．　10．　30）

Homomorphism Theorem of Generalized Logic Programs

Kiyoshi Akama

Faculty of Engineering, Hokkaido University

In order to solve complicated problems, we often simplify the problem into more manageable abstract problems, and get information for the original problem by solving it in an abstract form. Such techniques are widely used in problem solving (**hierarchical planning**) or program analysis (**abstract interpretation**).
In this paper we propose a new framework and the conditions where such techniques are safely used. Based on the **theory of generalized logic programs**, we introduce the concept of **homomorphism** between two program domains and prove the **homomorphism theorem** which gives the relation between the **declarative semantics** of two programs. The theory is elegant and general because the theory of generalized logic programs unifies the theory of declarative semantics for various declarative programs, even if the program domain is concrete or **abstract**.

一般化論理プログラムの準同型定理

赤間 清

北海道大学　工学部　情報工学科

複雑な問題を解くために、もとの問題を扱いやすい形に抽象化して、その解からもとの問題に関する情報を得ることがよくある。そのような手法は問題解決（**階層的問題解決**）やプログラム解析（**抽象解釈**）などで、広く用いられている。
本論文では、そのような手法を議論するための基礎となる新しい枠組を提案する。我々は**一般化論理プログラムの理論**を基礎にして、２つのプログラム領域の間の**準同型写像**の概念を新しく導入し、２つのプログラムの**宣言的意味**に関する**準同型定理**を証明する。ここで一般的でエレガントな理論が得られるのは、一般化論理プログラムの理論が広範な宣言的プログラムに対する統一的な宣言的意味論を与えているためで、それによって具体的な領域だけでなく、**抽象的な領域**のプログラムもまったく同様に扱うことができるからである。

1 Introduction

When we are asked whether `25676 + 10851` is even or odd, we can immediately answer that the result is odd. Instead of performing the addition of `25676` and `10851`, we use the rule of modulo arithmetic that tells us that adding a even number with an odd number always gives us an odd result. We use such techniques very often to solve complicated problems in the field of computer science and artificial intelligence [6, 5]. Therefore it is very useful to find the conditions where such techniques can be used safely.

In this paper we formalize the safety conditions in the framework of the theory of generalized logic programs [1, 3]. For this purpose, we will discuss the following items.

- two **specialization systems**, which correspond to, respectively, the concrete base domain and the abstract one.
- two **generalized logic programs**, which correspond to, respectively, the concrete system of computation and the abstract one.
- a **homomorphism** between two specialization systems, which explains the reason why we can regard the one base domain as concrete and the other as abstract.
- a **homomorphic relation** between two generalized logic programs, which explains the reason why we can regard the one system of computation as concrete and the other as abstract
- **declarative semantics** of two generalized logic programs, which correspond to the meaning of, respectively, the concrete system of computation and the abstract one.
- a **inclusion relation** between the two declarative semantics, which guarantees that the abstract system of computation can be safely used for getting information of the result of the concrete computation.

Note that such a theory can not be constructed elegantly in the usual framework of logic programs, because the usual concept of logic programs is too specific to deal with both concrete and abstract programs uniformly. In the usual theory the base domain is fixed to the usual concrete atoms consisting of predicate and terms, which provides the foundation for discussing concrete programs. However it does not provide a base domain for abstract programs. Strictly speaking, in the usual theory abstract programs are not logic programs in the sense that they do not have the same declarative semantics as the concrete programs have. This difficulty is overcome by the theory of generalized logic programs, because it provides us a very general concept of logic programs on various base domains, whether concrete or abstract.

2 Specialization System and Programs

2.1 Specialization Systems

By generalizing the structure of terms and substitutions, we have defined [1] more abstract structures called specialization systems, which are the formalization of base domains on which generalized logic programs (GLPs) are defined.

Definition 1 A *specialization system* is a 4-tuple $< \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu >$ that satisfies the following conditions.

(1) $\mu : \mathcal{S} \rightarrow partial\_map(\mathcal{A})$

(2) $\forall s_1, s_2 \in \mathcal{S}, \exists s \in \mathcal{S} : \mu(s) = \mu(s_2) \circ \mu(s_1)$

(3) $\exists s \in \mathcal{S}, \forall a \in \mathcal{A} : \mu(s)(a) = a$

(4) $\mathcal{A} \supset \mathcal{G}$

Elements of $\mathcal{A}$ are called objects or atoms. Elements of $\mathcal{S}$ are called *specializations*. The specializations that satisfy (3) are called *identity specializations*.

When there is no danger of confusion, we regard elements in $\mathcal{S}$ as partial mappings over $\mathcal{A}$, and use the following notational convention. Each element in $\mathcal{S}$ which is identified as a partial mapping on $\mathcal{A}$ is denoted by a Greek letter such as θ, and the application of such a partial mapping is represented by postfix notation. For example, $s \in \mathcal{S}, \mu(s)(a)$ and $\mu(s_n) \circ \mu(s_{n-1}) \circ \cdots \circ \mu(s_1)$ are denoted respectively by $\theta \in \mathcal{S}, a\theta$ and $\theta_1 \circ \theta_2 \circ \cdots \circ \theta_n$. The composition operator $\circ$ is often omited.

From the definition, μ is a subset of $\mathcal{S} \times partial\_map(\mathcal{A})$. As each element in $partial\_map(\mathcal{A})$ is a subset of $\mathcal{A} \times \mathcal{A}$, μ is a subset of $\mathcal{S} \times powerset(\mathcal{A} \times \mathcal{A})$. We often regard μ as a subset of $\mathcal{S} \times \mathcal{A} \times \mathcal{A}$, because $\mu \subset \mathcal{S} \times powerset(\mathcal{A} \times \mathcal{A})$ and $\nu \subset \mathcal{S} \times \mathcal{A} \times \mathcal{A}$ determine each other uniquely by the following equations.

$$\mu = \{(s, M) \mid M = \{(a, b) \mid (s, a, b) \in \nu\}\}$$
$$\nu = \{(s, a, b) \mid (s, M) \in \mu, (a, b) \in M\}$$

2.2 Examples of Specialization Systems

Throughout this paper we use the example of modulo arithmetic. Here we will give two examples of specialization systems which correspond to concrete and abstract base domains for the example of modulo arithmetic.

Example 1 [a specialization system for the concrete system of computation] We use usual terms to represent concrete numbers. Let $V_1 = \{\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{W}, \mathtt{U}, \cdots\}$, $K_1 = \{\mathtt{0}\}$, $F_1 = \{\mathtt{suc}\}$ and $P_1 = \{\mathtt{plus}\}$. Each element of V_1, K_1, F_1 and P_1 is called, respectively, a *variable*, a *constant*, a *function* and a *predicate*. Let $\mathcal{T}_1$ be the set of all terms (in the usual sense) over V_1, K_1 and F_1. For instance, `0`, `suc(suc(X))` and `suc(0)` are terms over V_1, K_1 and F_1. Let $\mathcal{A}_1$ be the set of all atoms of the form $\mathtt{plus}(t_1, t_2, t_3)$ where t_i's are terms in $\mathcal{T}_1$. For instance, `plus(0,suc(suc(X)),suc(0))` is an atom in $\mathcal{A}_1$. Let $\mathcal{G}_1$ be the set of all ground atoms in $\mathcal{A}_1$, that is, $\{\mathtt{plus}(t_1, t_2, t_3) \mid t_1, t_2, t_3 \in \{\mathtt{0}, \mathtt{suc(0)}, \mathtt{suc(suc(0))}, \cdots\}\}$. Let $\mathcal{S}_1$ be the set of all substitutions over V_1 and $\mathcal{T}_1$. Substitutions in $\mathcal{S}_1$ are sets of pairs of variables in V_1 and terms in $\mathcal{T}_1$. For instance, `{X/suc(Y),Y/0}` is a substitution. Application of a substitution θ to atoms defines a mapping M_θ on $\mathcal{A}_1$. The mapping, $\mu_1 : \mathcal{S}_1 \rightarrow map(\mathcal{A}_1)$ is also defined to give such a mapping M_θ for each substitution θ. For instance,

$$\mu_1(\{\mathtt{X/suc(0)}\})(\mathtt{plus(0, suc(suc(X)), suc(X))})$$
$$= \mathtt{plus(0, suc(suc(suc(0))), suc(suc(0)))}.$$

Then the 4-tuple $\Gamma_1 =< \mathcal{A}_1, \mathcal{G}_2, \mathcal{S}_1, \mu_1 >$ is a specialization system.

Example 2 [a specialization system for the abstract system of computation] We use "abstract terms" to represent "abstract numbers". Let $V_2 = V_1$, $K_2 = \{\mathtt{even}, \mathtt{odd}\}$, $F_2 = F_1$ and $P_2 = P_1$. Each element of V_2, K_2, F_2 and P_2 is called, respectively, a *variable*, a *constant*, a *function* and a *predicate*. Let $\mathcal{T}_2 = V_2 \cup K_2 \cup \{\mathtt{suc}(x) \mid x \in V_2\}$ be a set of terms over V_2, K_2 and F_2. For instance, `Y`, `even`, `odd`, `suc(X)` and `suc(Y)` are terms in $\mathcal{T}_2$. Note that `suc(suc(X))` and `suc(suc(suc(even)))` are not terms in $\mathcal{T}_2$. Let $\mathcal{A}_2$ be the set of all atoms of the form $\mathtt{plus}(t_1, t_2, t_3)$ where t_i's are terms in $\mathcal{T}_2$. For instance, `plus(odd, suc(X), even)` is an atom in $\mathcal{A}_2$. Note that `plus(odd, suc(suc(X)), suc(odd))` is not an atom in $\mathcal{A}_2$ because `suc(suc(X))` and `suc(odd)` are not in $\mathcal{T}_2$. Let $\mathcal{G}_2$ be the set of all ground atoms in $\mathcal{A}_2$, that is, $\mathcal{G}_2 = \{\mathtt{plus}(t_1, t_2, t_3) \mid t_1, t_2, t_3 \in \{\mathtt{even}, \mathtt{odd}\}\}$. Let $\mathcal{S}_2$ be the set of all substitutions over V_2 and $\mathcal{T}_2$. Substitutions in $\mathcal{S}_2$ are sets of pairs of variables in V_2 and terms in $\mathcal{T}_2$. For instance, `{X/suc(Y), Y/even, Z/W}` is a substitution in $\mathcal{S}_2$. But `{X/suc(suc(Y)),Y/0}` is not a substitution in $\mathcal{S}_2$ because `suc(suc(Y))` and `0` are not in $\mathcal{T}_2$. Next we define a mapping $\nu_2 : \mathcal{S}_2 \rightarrow map(\mathcal{T}_2)$ as follows.

$$
\begin{array}{ll}
\nu_2(\theta)(k) = k & \cdots \quad \theta \in \mathcal{S}_2, k \in K_2 \\
\nu_2(\theta)(x) = x\theta & \cdots \quad \theta \in \mathcal{S}_2, x \in V_2 \\
\nu_2(\theta)(\mathtt{suc}(x)) = \mathtt{suc}(x\theta) & \cdots \quad \theta \in \mathcal{S}_2, x \in V_2, x\theta \in V_2 \\
\nu_2(\theta)(\mathtt{suc}(x)) = \mathtt{odd} & \cdots \quad \theta \in \mathcal{S}_2, x \in V_2, x\theta = \mathtt{even} \in K_2 \\
\nu_2(\theta)(\mathtt{suc}(x)) = \mathtt{even} & \cdots \quad \theta \in \mathcal{S}_2, x \in V_2, x\theta = \mathtt{odd} \in K_2 \\
\nu_2(\theta)(\mathtt{suc}(x)) = y & \cdots \quad \theta \in \mathcal{S}_2, x \in V_2, x\theta = \mathtt{suc}(y), y \in V_2
\end{array}
$$

For instance, $\nu_2(\{\mathtt{X/even}\})(\mathtt{odd}) = \mathtt{odd}$, $\nu_2(\{\mathtt{X/even}\})(\mathtt{X}) = \mathtt{even}$, $\nu_2(\{\mathtt{X/even}\})(\mathtt{suc(Y)}) = \mathtt{suc(Y)}$, $\nu_2(\{\mathtt{X/even}\})(\mathtt{suc(X)}) = \mathtt{odd}$. Using ν_2 we define μ_2 by $\mu_2(\theta)(\mathtt{plus}(t_1, t_2, t_3)) = \mathtt{plus}(\nu_2(\theta)(t_1), \nu_2(\theta)(t_2), \nu_2(\theta)(t_3))$ where t_i's are terms in $\mathcal{T}_2$. For example,

$\mu_2(\{\mathtt{X/suc(Y)}, \mathtt{Y/even}\})(\mathtt{plus(even, suc(X), suc(Y))}) = \mathtt{plus(even, Y, odd)}$.

Then the 4-tuple $\Gamma_2 =< \mathcal{A}_2, \mathcal{G}_2, \mathcal{S}_2, \mu_2 >$ is a specialization system.

2.3 Programs on Specialization Systems

We review the definition of generalized logic programs (GLPs) on specialization systems. Details are found in [1]. We fix a specialization system $\Gamma =< \mathcal{A}, \mathcal{G}, \mathcal{S}, \mu >$.

Definition 2 A *program clause* on $\mathcal{A}$ (or on Γ) is a clause of the form : $H \leftarrow A_1, \cdots, A_n$, where $H, A_1, \cdots, A_n$ are atoms in $\mathcal{A}$. A *logic program* on $\mathcal{A}$ (or on Γ) is a (possibly infinite) set of program clauses on $\mathcal{A}$ (or on Γ).

The set of all program clauses on $\mathcal{A}$ (or on Γ) is denoted by $Pclause(\mathcal{A})$ (or by $Pclause(\Gamma)$). The set of all programs on $\mathcal{A}$ (or on Γ) is denoted by $Program(\mathcal{A})$ (or by $Program(\Gamma)$).

Example 3 [a GLP corresponding to the concrete system of computation] Let P_1 be a set $\{C_{11}, C_{12}\}$ of two clauses:

C_{11}: `plus(0,Y,Y)` $\leftarrow$

C_{12}: `plus(suc(X),Y,suc(Z))` $\leftarrow$ `plus(X,Y,Z)`

Then P_1 is obviously a program on Γ_1.

Example 4 [a GLP corresponding to the abstract system of computation]
Let P_2 be a set $\{C_{21}, C_{22}\}$ of two clauses:

C_{21}: `plus(even,Y,Y)` $\leftarrow$

C_{22}: `plus(suc(X),Y,suc(Z))` $\leftarrow$ `plus(X,Y,Z)`

Then P_2 is obviously a program on Γ_2.

3 Homomorphism

3.1 Tuples of Mappings

Let $D_i (1 \leq i \leq n)$ and $R_i (1 \leq i \leq n)$ be sets. The set of all tuples $< h_1, h_2, \cdots, h_n >$ of mappings $h_i : D_i \rightarrow R_i (1 \leq i \leq n)$ is denoted by $Map(D_1, R_1; D_2, R_2; \cdots; D_n, R_n)$.

Tuples of mappings determine various mappings. For example, let $h =< h_1, h_2 >$ be a tuple of mappings in $Map(D_1, R_1; D_2, R_2)$. Let $\gamma : D_1 \times D_2 \rightarrow R_1 \times R_2$ be defined by $\gamma(x, y) = (h_1(x), h_2(y))$. We write $h(x, y)$ instead of $\gamma(x, y)$ only when its meaning is obvious from the contexts. Such convention is also used in the case of h_1 and h_2. We write $h(x)$ instead of $h_1(x)$, and $h(y)$ instead of $h_2(y)$ as far as we can understand the meaning. Such convention is widely used in this paper because of simplicity.

Example 5 [a tuple of mappings]
We give an example of a tuple of mappings, which will turn out to be a homomorphism in example 6. Let us define $h =< h_{\mathcal{A}}, h_{\mathcal{S}} >$ in $Map(\mathcal{A}_1, \mathcal{A}_2; \mathcal{S}_1, \mathcal{S}_2)$. First we introduce the following notational convention. Terms of the form `E`, `suc(E)`, `suc(suc(E))`, `suc(suc(suc(E)))`, `suc(suc(suc(suc(E))))`, $\cdots$ are denoted, respectively, by $\mathtt{suc}^0(\mathtt{E})$, $\mathtt{suc}^1(\mathtt{E})$, $\mathtt{suc}^2(\mathtt{E})$, $\mathtt{suc}^3(\mathtt{E})$, $\mathtt{suc}^4(\mathtt{E})$, $\cdots$.

Note that $\mathcal{T}_1 = \{\mathtt{suc}^n(0) \mid n \in \{0, 1, 2, \cdots\}\} \cup \{\mathtt{suc}^n(x) \mid n \in \{0, 1, 2, \cdots\}, x \in V_1\}$. Therefore we can define $h_{\mathcal{T}} : \mathcal{T}_1 \rightarrow \mathcal{T}_2$ by

$$
\begin{array}{lll}
h_{\mathcal{T}}(\mathtt{suc}^n(0)) = \mathtt{even} & \cdots & n \text{ is even.} \\
h_{\mathcal{T}}(\mathtt{suc}^n(0)) = \mathtt{odd} & \cdots & n \text{ is odd.} \\
h_{\mathcal{T}}(\mathtt{suc}^n(x)) = x & \cdots & n \text{ is even, } x \in V_1 \\
h_{\mathcal{T}}(\mathtt{suc}^n(x)) = \mathtt{suc}(x) & \cdots & n \text{ is odd, } x \in V_1
\end{array}
$$

This definition is well defined since $V_1 = V_2$.

Now we can define $h =< h_{\mathcal{A}}, h_{\mathcal{S}} >$ using $h_{\mathcal{T}}$.

- $h_{\mathcal{A}} : \mathcal{A}_1 \rightarrow \mathcal{A}_2$ by $h_{\mathcal{A}}(\mathtt{plus}(t_1, t_2, t_3)) = \mathtt{plus}(h_{\mathcal{T}}(t_1), h_{\mathcal{T}}(t_2), h_{\mathcal{T}}(t_3))$.
- $h_{\mathcal{S}} : \mathcal{S}_1 \rightarrow \mathcal{S}_2$ by $h_{\mathcal{S}}(\theta) = \{x/h_{\mathcal{T}}(t) \mid x/t \in \theta, x \in V_1, t \in \mathcal{T}_1\}$

These are well defined because $P_1 = P_2 = \{\mathtt{plus}\}$, $h_{\mathcal{T}}$ is from $\mathcal{T}_1$ to $\mathcal{T}_2$ and $V_1 = V_2$.
Then, $h =< h_{\mathcal{A}}, h_{\mathcal{S}} >$ is a tuple of mappings in $Map(\mathcal{A}_1, \mathcal{A}_2; \mathcal{S}_1, \mathcal{S}_2)$.

3.2 Homomorphism

We define a homomorphism from a specialization system to a specialization system.

Definition 3 Let $\Gamma_1 =< \mathcal{A}_1, \mathcal{G}_1, \mathcal{S}_1, \mu_1 >$ and $\Gamma_2 =< \mathcal{A}_2, \mathcal{G}_2, \mathcal{S}_2, \mu_2 >$ be specialization systems. Let h be a tuple of mappings in $Map(\mathcal{A}_1, \mathcal{A}_2; \mathcal{S}_1, \mathcal{S}_2)$. h is a *homomorphism* from Γ_1 to Γ_2 iff (1) $h(\mu_1) \subset \mu_2$ and (2) $h(\mathcal{G}_1) \subset \mathcal{G}_2$.

Note that μ_1 in $h(\mu_1)$ and μ_2 in $h(\mu_2)$ are regarded respectively as subsets of $\mathcal{S}_1 \times \mathcal{A}_1 \times \mathcal{A}_1$ and $\mathcal{S}_2 \times \mathcal{A}_2 \times \mathcal{A}_2$. Note also that h in definition 3 is regarded as the following four kind of mappings :

1. a mapping form $\mathcal{A}_1$ to $\mathcal{A}_2$
2. a mapping from $\mathcal{S}_1$ to $\mathcal{S}_2$
3. a mapping from $powerset(\mathcal{S}_1 \times \mathcal{A}_1 \times \mathcal{A}_1)$ to $powerset(\mathcal{S}_2 \times \mathcal{A}_2 \times \mathcal{A}_2)$
4. a mapping from $powerset(\mathcal{A}_1)$ to $powerset(\mathcal{A}_2)$

Proposition 1 Let h be a homomorphism from Γ_1 to Γ_2. If an element θ in $\mathcal{S}_1$ is applicable to an element a in $\mathcal{A}_1$, then

(1) $h(\theta)$ is applicable to $h(a)$, and

(2) $h(a\theta) = h(a)h(\theta)$

Example 6 We prove that $h =< h_{\mathcal{A}}, h_{\mathcal{S}} >$ is a homomorphism from Γ_1 and Γ_2.

1. [**Proof of** $h(\mu_1) \subset \mu_2$]
   $h(\theta, \mathsf{suc}^m(0), \mathsf{suc}^m(0)) = (h(\theta), \mathsf{odd}, \mathsf{odd}) \in \mu_2 \cdots m$ is odd.
   $h(\theta, \mathsf{suc}^m(0), \mathsf{suc}^m(0)) = (h(\theta), \mathsf{even}, \mathsf{even}) \in \mu_2 \cdots m$ is even.
   $h(\theta, \mathsf{suc}^m(x), \mathsf{suc}^m(x)) = (h(\theta), \mathsf{suc}(x), \mathsf{suc}(x)) \in \mu_2 \cdots x \in V_1,\ x/t \notin \theta,\ m$ is odd.
   $h(\theta, \mathsf{suc}^m(x), \mathsf{suc}^m(x)) = (h(\theta), x, x) \in \mu_2 \cdots x \in V_1,\ x/t \notin \theta,\ m$ is even.
   $h(\theta, \mathsf{suc}^m(x), \mathsf{suc}^{m+n}(y)) = (h(\theta), \mathsf{suc}(x), y) \in \mu_2$
   $\cdots x \in V_1,\ x/\mathsf{suc}^n(y) \in \theta,\ y \in V_1,\ m$ is odd, n is odd.
   Other cases are omitted because of space limitation.

2. [**Proof of** $h(\mathcal{G}_1) \subset \mathcal{G}_2$]
   Since $\mathcal{G}_1 = \{\mathsf{suc}^m(0) \mid m \in \{0, 1, 2, \cdots\}\}$, $h(\mathcal{G}_1) = \{\mathsf{even}, \mathsf{odd}\}$, which is equal to $\mathcal{G}_2$.

3.3 Program Transformation by Homomorphism

A mapping $h : \mathcal{A}_1 \rightarrow \mathcal{A}_2$ is naturally extended to a mapping $h : Pclause(\mathcal{A}_1) \rightarrow Pclause(\mathcal{A}_2)$, which maps a clause $C = (H \leftarrow B_1, B_2, \cdots, B_n)$ on $\mathcal{A}_1$ to a clause $h(C) = (h(H) \leftarrow h(B_1), h(B_2), \cdots, h(B_n))$ on $\mathcal{A}_2$.

Moreover, h is extended to a mapping $h : Program(\Gamma_1) \rightarrow Program(\Gamma_2)$ which maps a program P on Γ_1 to a program $h(P) = \{h(C) \mid C \in P\}$ on Γ_2.

Example 7 We have already defined $h =< h_{\mathcal{A}}, h_{\mathcal{S}} >$ in example 5, which is a homomorphism from Γ_1 to Γ_2. We have also defined P_1 and P_2 in example 3 and 4, which are programs, respectively, on Γ_1 and Γ_2. Then the first clause C_{11} in P_1 : (`plus(0,Y,Y)` $\leftarrow$) is transformed by h into the first clause C_{21} in P_2 : (`plus(even,Y,Y)` $\leftarrow$) because h(`0`) = `even`. The second clause C_{12} of P_1 : (`plus(suc(X),Y,suc(Z))` $\leftarrow$ `plus(X,Y,Z)`) is transformed by h into the second clause C_{22} of P_2 : (`plus(suc(X),Y,suc(Z))` $\leftarrow$ `plus(X,Y,Z)`), which is identical to C_{12}. Therefore, the program P_1 on Γ_1 is transformed by h into the program P_2 on Γ_2, that is, $h(P_1) = P_2$.

4 Homomorphism Theorem

The aim of this section is to establish the relation between the semantics of two programs which are connected by a homomorphism of specialization systems.

4.1 Declarative Semantics and Homomorphism

The semantics of a program P, which is denoted by $\mathcal{M}(P)$, is defined by

$$\mathcal{M}(P) = M_P = lfp(K_P) = K_P \uparrow \omega = lfp(T_P) = T_P \uparrow \omega = \cup[T_P]^n(\emptyset).$$

where T_P is a one-step-inference transformation of P and $K_P = T_P + I_d$ is a knowledge-increasing transformation of P [2]. In the following, $[T_P]^n(\emptyset)$ is denoted by $T(P,n)$.

We review the proposition which explains the relation between $T(P,n)$ and $\mathcal{M}(P)$.

Proposition 2 Let P be a logic program on a specialization system Γ. Then,

$$g \in \mathcal{M}(P) \Leftrightarrow \exists n \in \{1,2,3,\cdots\} : g \in T(P,n)$$

Example 8 We give the declarative semantics of logic programs P_1 and P_2 in example 3 and 4.

$$\begin{aligned}\mathcal{M}(P_1) = \{\ & \mathtt{plus}(\mathtt{suc}^m(0), \mathtt{suc}^n(0), \mathtt{suc}^{m+n}(0)) \\ & \mid m \in \{0,1,2,\cdots,\}, n \in \{0,1,2,\cdots,\}\ \}. \\ \mathcal{M}(P_2) = \{\ & \mathtt{plus(even,\ even,\ even)},\ \mathtt{plus(even,\ odd,\ odd)}, \\ & \mathtt{plus(odd,\ even,\ odd)},\ \mathtt{plus(odd,\ odd,\ even)}\ \}\end{aligned}$$

4.2 Homomorphism Theorem

First we give a proposition which relates a homomorphism h with $T(P,n)$.

Proposition 3 Let h be a homomorphism from Γ_1 to Γ_2. Let P be a logic program on Γ_1. Then [1] , for arbitrary non-negative integer n,

$$h(T(P,n)) \subset T(h(P),n).$$

The following is the main theorem in this paper.

[1] $T(P,n)$ is a subset of $\mathcal{G}_1$. h in $h(T(P,n))$ is a mapping from $powerset(\mathcal{G}_1)$ to $powerset(\mathcal{G}_2)$. Then, $h(T(P,n)) \subset \mathcal{G}_2$.

Theorem 1 Let h be a homomorphism from Γ_1 to Γ_2. For any logic program P on Γ_1,

$$h(\mathcal{M}(P)) \subset \mathcal{M}(h(P))$$

Example 9 As we have discussed in example 7 and 8:

$$\mathcal{M}(P_1) = \{\ \mathtt{plus}(\mathtt{suc}^m(0), \mathtt{suc}^n(0), \mathtt{suc}^{m+n}(0)) \mid m \in \{0,1,2,\cdots\}, n \in \{0,1,2,\cdots\}\ \}.$$

$$\mathcal{M}(h(P_1)) = \{\ \mathtt{plus(even,\ even,\ even)},\ \mathtt{plus(even,\ odd,\ odd)},\ \mathtt{plus(odd,\ even,\ odd)},\ \mathtt{plus(odd,\ odd,\ even)}\ \}$$

And

$h_{\mathcal{T}}(\mathtt{suc}^n(0)) = \mathtt{even} \quad \cdots \quad n$ is even.

$h_{\mathcal{T}}(\mathtt{suc}^n(0)) = \mathtt{odd} \quad \cdots \quad n$ is odd.

$h_{\mathcal{A}}(\mathtt{plus}(t_1,t_2,t_3)) = \mathtt{plus}(h_{\mathcal{T}}(t_1), h_{\mathcal{T}}(t_2), h_{\mathcal{T}}(t_3))$

Since $(m+n) \bmod 2 = ((m \bmod 2) + (n \bmod 2)) \bmod 2$, it is obvious that $h(\mathcal{M}(P_1)) = h_{\mathcal{A}}(\mathcal{M}(P_1)) = \mathcal{M}(h(P_1))$.

5 Concluding Remarks

We have introduced the concept of **homomorphism** between two specialization systems and proved the **homomorphism theorem**. This is a very general and elegant theory for discussing the relation between the declarative semantics of two generalized logic programs on (possibly different) specialization systems. In the examples here $h(\mathcal{M}(P_1))$ is equal to $\mathcal{M}(h(P_1))$, but in general $h(\mathcal{M}(P_1))$ is a subset of $\mathcal{M}(h(P_1))$. An example where $h(\mathcal{M}(P_1))$ is not equal to $\mathcal{M}(h(P_1))$ is given in [4]. An isomorphism theorem is discussed in [4]. All the proofs of propositions and theorems in this paper are also found there. This theory is useful for **hierarchical planning** and **abstract interpretation**. Application to planning has been discussed in [7].

References

[1] Akama,K. : Generalized Logic Programs on Specialization Systems, *Preprints Work. Gr. for Programming, IPSJ* 6-4-PRG, pp.31-40 (1992)

[2] Akama,K. : Least Fixpoint Semantics of Generalized Logic Programs, *Preprints Work. Gr. for Artificial Intelligence, IPSJ* 83-2-AI, pp.33-42 (1992)

[3] Akama,K. : Unfolding of Generalized Logic Programs, *Preprints Work. Gr. for Artificial Intelligence, IPSJ* 83-3-AI, pp.43-52 (1992)

[4] Akama, K. : Homomorphism Theorem of Generalized Logic Programs, *Hokkaido University Information Engineering Technical Report*, HIER-LI-9217 (1992)

[5] Cousot, P. and Cousot, R. : Abstract Interpretation and Application to Logic Programs, *J. Logic Programming*, Vol.13 No.2&3, pp.103-179 (1992)

[6] Knoblock, C.A. : A Theory of Abstraction for Hierarchical Planning, *Proceedings of the Workshop on Change of Representation and Inductive Bias*, Kluwer, Boston, MA, (1989)

[7] Mabuchi, H., Tsurutani, T., Akama, K. and Miyamoto,E.: Planning based on homomorphism, *Proc. of JSAI'92* (in Japanese) 12-1, pp.465-468 (1992)