数理解析研究所講究録
第 655 巻 1988 年 104-123

On Operational Semantics of Congruence Relation Defined in Algebraic Language ASL/*

Hiroyuki SEKI, Kenichi TANIGUCHI and Tadao KASAMI
(関 浩之 谷口 健一 嵩 忠雄)

Department of Information and Computer Sciences
Faculty of Engineering Science
Osaka University
Toyonaka, Osaka 560, JAPAN

Abstract An algebraic specification (or text) specifies a congruence relation on a set of expressions. In algebraic language ASL/*, a pair *(G, AX)* is called a *text*, where $G$ is a context-free grammar and $AX$ is a set of axioms. A text $t=(G, AX)$ specifies the set E_G of expressions generated by $G$ and the least congruence relation on E_G satisfying all the axioms in $AX$, where 'congruency' is defined based on the syntax (phrase structure) of the expressions. In general, for a text $t$ in ASL/*, the condition, (A) $e$ is congruent with e' in $t$, is not logically equivalent to the condition, (B) e' is obtained from $e$ by rewriting $e$ when the axioms in $t$ are regarded as 'bidirectional' rewrite rules. We present a sufficient condition for a text $t$ under which (A) and (B) are equivalent for any pair of expressions $e$ and e', which means that the congruence relation specified by $t$ is simply defined operationally.

1. Introduction

In a stepwise refinement process of a software development activity, it is desirable that the semantics of programs and their specifications should be defined formally. Algebraic semantics is one of the most promising methods to define the semantics of programs and specifications, and has the following advantages [1]:

(1) Since the semantics of both of the programs and their specifications (called *texts*, in this paper) are defined simply and clearly by using congruence relation, the correctness of a text can be verified relatively easily.
(2) A text can be written at an arbitrary level of abstraction; a text corresponding to a specification written in a natural language can be refined stepwise to obtain an efficient program satisfying the given specification in the single semantical framework. Especially, a class of functional programs can be regarded as a special subclass of texts [5].

On the other hand, it has been pointed out from the practical point of view that several problems arise when defining a fairly large specification in algebraic methods. They are summarized as follows:
(a) The syntax of the terms (or *expressions* in this paper) is usually restricted to so called prefix notation such as $p(g(c_1, c_2), c_3)$, and it is not allowed for the writers to define arbitrary syntax, such as infix notation, at their own discretion.
(b) Different data types (or sorts) cannot share common syntax. Hence, if there is an inclusion relation among sorts (or data types, e.g., integer and real), (1) the definition of each operation on each type must be given separately even if they are homomorphic (e.g., the addition '+' on integer and '+' on real), and, (2) the type transformation must be specified explicitly ('1' on integer vs. '1.0' on real).
(c) When error handling and/or exception handling are considered (e.g., popping the empty stack results in the error state and no operation is defined on that error state), the text tends to be complicated [1].

In order to resolve these problems, several extensions of algebraic semantics such as *error algebra*[2] have been proposed. However, the semantics can no longer be defined simply in these approaches.

In ASL/*, the syntax of expressions can be de[illegible]ed by using context-free grammar (abbreviated as *cfg*). An inclusion relation among sorts can be represented by an inclusion relation among the sets of expressions which are derived from specific non-terminal symbols corresponding to the sort names. Error handling can be also simply specified as follows: First, let G be a cfg which has two specific non-terminal symbols; (a) a non-terminal symbol from which all the states including both legal and error states are derived, and, (b) a non-terminal symbol, say *state,* from which only the legal states are derived. Then, a 'strict' operation should be defined only on the expressions derived from the non-terminal symbol *state.* Details of sort inclusion and error handling are described in [6, 8].

Since the set of expressions is defined by a cfg in ASL/*, the concept 'congruency' must be generalized so that a congruence relation can be defined on a context-free language (*cfl*). Remember that a binary relation R is called a congruence relation if (1) R is an equality relation, i.e., R is reflexive, symmetric and transitive, and (2) R is closed under each operation, i.e., for each (n-ary) operation f, whenever $e_1 R e_1'$, $e_2 R e_2'$, ..., $e_n R e_n'$ hold then $f(e_1, e_2, ..., e_n) R f(e_1', e_2', ..., e_n')$ also holds. In ASL/*, for each expression e, several substrings of e are specified as *subexpressions*, which are considered to be arguments of the operation represented by e. A congruence relation is defined to be an equality relation which is closed under replacement of subexpression.

In ASL/*, a pair (G, AX) is called a *text*, where G is a cfg without a start symbol, and AX is a finite set of axioms. A text $t=(G, AX)$ specifies the least congruence relation on E_G which satisfies all the axioms in AX, where E_G is the set of all the expressions derived from some non-terminal symbol in G. A string in E_G is called an *expression*. More formally, for an axiom $l==r$ in AX, let $Q_t(ax)$ denote $\{(l\theta, r\theta) \mid l\theta$ (or $r\theta$) is an arbitrary expression obtained from l (or r) by substituting an expression for each variable in l (or r)$\}$ and Q_t denote $\bigcup_{ax \in AX} Q_t(ax)$. *The congruence relation* $\bar{Q}_t$ *specified by the text* t is defined to be the least congruence relation on E_G containing Q_t. For expressions e and e', if $e\ \bar{Q}_t\ e'$ holds then e is said to be *congruent with* e' *(in t)*.

In the case that the syntax of expressions is restricted to the prefix notation, the close relation between algebraic axioms and rewriting systems is well-known [4], i.e., the following property holds in our terminology.
[Operational Completeness (text version)] For a given text $t=(G, AX)$, e is congruent with e' if and only if e' is obtained from e by rewriting e finitely many times when Q_t is regarded as a set of 'bidirectional' rewrite rules. ■

Unfortunately, in ASL/*, operational completeness does not hold in general. In this paper, a sufficient condition for a given text t in ASL/* is presented under which operational completeness stated above holds so that the semantics of the congruence relation specified by t is simply defined operationally.

In section 2, for a set E of expressions and a relation R on E (R is called a *set of equations*), *the congruence relation* $\bar{R}$ *on* E *generated by* R is defined in an abstract way; $\bar{R}$ is defined independently of the mechanisms which specify the set E of expressions and the set R of equations. Then, we give a sufficient condition for a set E of expressions and a set R of equations to satisfy operational completeness:

[Operational Completeness (abstract version)] For any expressions e and e' in E, $e \bar{R} e'$ holds if and only if e' is obtained from e by rewriting e finitely many times when R is regarded as a set of rewrite rules. ■

For a relation R, let R^{-1} be the inverse of R, i.e., $e\ R^{-1}\ e'$ if and only if $e'\ R\ e$. The sufficient condition for E and R is as follows.

[A Sufficient Condition for Operational Completeness (abstract version)]

(1) E satisfies the following *syntactic transitivity.*

[Syntactic Transitivity] For any expressions e_1, e_2 and e_3 in E, if e_1 is a subexpression of e_2 and e_2 is a subexpression of e_3, then e_1 is also a subexpression of e_3.

(2) E and R satisfy the following *local syntax compatibility.*

[Local Syntax Compatibility] For any pair (e_1, e_1') in $R \cup R^{-1}$ and any expression e_2 in E containing e_1 as a subexpression, let e_2' be the string obtained from e_2 by replacing e_1 with e_1'. Then e_2' is also an expression in E and e_1' is also a subexpression of e_2'. ■

In sections 3 and 4, for a text $t=(G, AX)$, we formally define the set of expressions E_G and the set Q_t of equations. Next, we assume that a cfg G satisfies a sufficient condition (*having unambiguous structure*, see 3.2) for G to satisfy syntactic transitivity. Then we give a decidable sufficient condition (called *global syntax compatibility*, see 3.3) for E_G and Q_t to satisfy local syntax compatibility. To summarize, for a given text $t=(G, AX)$, if (1) G has unambiguous structure and (2) Q_t satisfies global syntax compatibility (which is decidable under the assumption that (1) holds) then operational completeness (text version) holds.

2. A Congruence Relation on a Set of Expressions

2.1 Expressions and Subexpressions

Let V be a finite set of symbols. Let V^* denote the set of all the strings on V and let $V^+ = V^* - \{\varepsilon\}$, where ε is the empty string. The length of a string u is denoted by $|u|$. For strings u and w, if $w = v_1 u v_2$ for some v_1 and v_2 then u is called a *substring of w at occurrence* $|v_1|$, or simply, a *substring of w*. For strings w, u' and a substring u of w, let $w[u \leftarrow u', i]$ denote the string obtained from w by replacing u with u', i.e.,

$$w[u \leftarrow u', i] = v_1 u' v_2$$

where $i = |v_1|$. If there is no ambiguity, $w[u \leftarrow u', i]$ may be abbreviated as $w[u \leftarrow u']$ by omitting the occurrence i.

Let E be a subset of V^*. A string in E is called an *expression (in E)*, and E is called a *set of expressions (on V)*. Let Z denote the set of all the non-negative integers. Let $\ni$ be a 3-ary relation on $E \times E \times Z$ (If (e_1, e_2, i) satisfies $\ni$, then we write $e_1 \ni e_2$ *at* i) which satisfies (1) e_1 is a substring of e_2 at i whenever $e_2 \ni e_1$ *at* i, and (2) $e \ni e$ *at* 0 holds for any expression e. e_1 is called a *subexpression of* e_2 *at occurrence* i, or simply, a *subexpression of* e_2 if and only if $e_2 \ni e_1$ *at* i. $e_2 \ni e_1$ *at* i is abbreviated as $e_2 \ni e_1$ if there is no ambiguity. We also say e_2 contains e_1 as a subexpression if $e_2 \ni e_1$. The relation $\ni$ is called *the syntax relation on E*. We assume that, when a set E of expressions is introduced, a syntax relation on E is also introduced (either explicitly or implicitly). Section 3 describes how a set of expressions and a syntax relation on it are specified.

2.2 Definition of a Congruence Relation

Let E be a set of expressions. A subset R of $E \times E$ is called a binary relation on E and we write $e\ R\ e'$ when $(e, e') \in R$. A binary relation C on E is said to be a *congruence relation on E* if the following conditions (1) and (2) are satisfied.

(1) C is an equality relation. i.e., C is reflexive, symmetric and transitive relation.

(2) [Congruency] For expressions e_1, e_1' and e_2 in E such that $e_1\ C\ e_1'$ and $e_2 \ni e_1$,

if $e_2[e_1 \leftarrow e_1']$, denoted e_2', is in E and $e_2' \ni e_1'$ holds, then $e_2\ C\ e_2'$ also holds. ■

Suppose that $e_2 \ni e_1$, i.e., e_1 is a subexpression of e_2, and let $e_2' = e_2[e_1 \leftarrow e_1']$ (the string obtained from e_2 by replacing e_1 with e_1'). It does not necessarily follow that e_2' is an expression in E. Furthermore e_1' is not always a subexpression of e_2' even if $e_2' \in E$. Congruency means that for expressions e_1 and e_1' which are already known to be 'congruent with each other', and an expression e_2 which contains e_1 as a subexpression, e_2 and $e_2' = e_2[e_1 \leftarrow e_1']$ must also be congruent if e_2' is an expression in E and e_1' is a subexpression of e_2'.

Let R be a binary relation on E. R is sometimes called a *set of equations. The congruence relation on E generated by R* (denoted by $\bar{R}$) is defined to be the least congruence relation on E containing R. For any set R of equations, $\bar{R}$ always exists and is uniquely determined [6].

2.3 Operational Semantics of a Congruence Relation

In this subsection, for any set R of equations, the congruence relation $\bar{R}$ generated by R is characterized operationally. For a relation R, let $R^{\#}$ be the reflexive-transitive closure of $R \cup R^{-1}$.

[Definition 2.1] For a set E of expressions and a relation R on E, define the relations $R_0, R_1, \ldots, R_\infty$ as follows.

(1) $R_0 = R$

(2) For each i $(i \geq 1)$, $e_2\, R_i\, e_2'$ if and only if

there exist expressions e_1 and e_1' in E such that $e_1\, R_{i-1}^{\#}\, e_1'$, $e_2 \ni e_1$, $e_2' = e_2[e_1 \leftarrow e_1']$ is in E and $e_2' \ni e_1'$.

(3) $e\, R_\infty\, e'$ if and only if $e\, R_i\, e'$ for some i $(i \geq 1)$. ■

Intuitively, $e\, R_1\, e'$ means that, when pairs of expressions in the reflexive-transitive closure of $R \cup R^{-1}$ are regarded as rewrite rules, e' is obtained from e by one step rewriting. Similarly, $e\, R_i\, e'$ means that, when pairs of expressions in $R_{i-1}^{\#}$ are regarded as rewrite rules, e' is obtained from e by one step rewriting, and $e\, R_\infty\, e'$ means that e' is obtained from e by one step rewriting under such rewrite rules.

[Lemma 2.1] For a set E of expressions and a relation R on E, $\bar{R} = R_\infty^{\#} = R_\infty$. ■

Let us define the relation $\underset{R}{\rightarrow}$ as follows:

$e_2 \underset{R}{\rightarrow} e_2'$

if and only if

there exist expressions e_1 and e_1' in E such that $e_1\, R\, e_1'$, $e_2 \ni e_1$, $e_2' = e_2[e_1 \leftarrow e_1']$ is in E and $e_2' \ni e_1'$.

The difference between $\underset{R}{\rightarrow}$ and R_i is that only R is used as a set of rewrite rules in the definition of $\underset{R}{\rightarrow}$ while $R_{i-1}^{\#}$ (i.e., reflexive-transitive closure of $R_{i-1} \cup R_{i-1}^{-1}$) is used in the definition of R_i. It is well-known that, if the syntax of expressions is restricted to the prefix form such as $f(e_1, e_2, \ldots, e_n)$, then $\bar{R} = (\underset{R}{\rightarrow})^{\#}$ holds [4], which means that the operational semantics of the congruence relation $\bar{R}$ can be defined simply. However $R_\infty^{\#}$ does not coincide with $(\underset{R}{\rightarrow})^{\#}$ in general. The reasons are as follows.

(1) Even if e_1 is a subexpression of e_2 and e_2 is a subexpression of e_3, e_1 is not always a subexpression of e_3. For example, assume that $e_2 \ni e_1$ and $e_3 \ni e_2$ hold but $e_3 \ni e_1$ does not hold. Let $e_2' = e_2[e_1 \leftarrow e_1']$ and $e_3' = e_3[e_2 \leftarrow e_2']$ and assume that $e_2' \ni e_1'$ and $e_3' \ni e_2'$. Let $R = \{ (e_1, e_1') \}$. Then, since $e_2 \ni e_1$ and $e_2' \ni e_1'$ hold by the assumption, $e_2\, R_1\, e_2'$ holds by definition 2.1. Similarly $e_3\, R_2\, e_3'$ and hence $e_3\, R_\infty^{\#}\, e_3'$ hold since $e_3 \ni e_2$ and $e_3' \ni e_2'$. On the other hand, since e_1 is

not a subexpression of e_3, (e_1, e_1') cannot be applied to e_3 as a rewrite rule. Similarly, (e_1', e_1) cannot be applied to e_3'. It follows that $e_3 (\underset{R}{\rightarrow})^{\#} e_3'$ does not hold and $R_\infty^{\#}$ does not coincide $(\underset{R}{\rightarrow})^{\#}$.

(2) For expressions e_1 and e_2 satisfying $e_2 \ni e_1$, $e_2' = e_2[e_1 \leftarrow e_1'] \ni e_1'$ does not always hold even if e_2' is an expression. For example, let $R = \{ (e_1, e_1'), (e_1', e_1'') \}$. Suppose $e_2 \ni e_1$ and let $e_2' = e_2[e_1 \leftarrow e_1']$ and $e_2'' = e_2[e_1 \leftarrow e_1'']$. Suppose that $e_2'' \ni e_1''$ but e_1' is not a subexpression of e_2'. $e_2 R_1^{\#} e_2''$ and $e_2 R_\infty^{\#} e_2''$ hold since $e_2 \ni e_1$, $e_2'' \ni e_1''$ and $e_1 R^{\#} e_1''$ (by the definition of R). On the other hand, e_1' is not a subexpression of e_2' and so e_2'' (or e_2) cannot be obtained from e_2 (or e_2'') by rewriting under rewrite rules in $R \cup R^{-1}$. Hence $e_2 (\underset{R}{\rightarrow})^{\#} e_2''$ does not hold and $R_\infty^{\#}$ does not coincide with $(\underset{R}{\rightarrow})^{\#}$. ■

Suppose that a set E of expressions satisfies the following condition.

[Syntactic Transitivity] For any expressions e_1, e_2 and e_3, if $e_2 \ni e_1$ and $e_3 \ni e_2$, then $e_3 \ni e_1$. ■

Then, in the example in (1) above, $e_3 \ni e_1$ and $e_3' \ni e_1'$ hold. Hence $(e_1, e_1') \in R$ can be applied to e_3 as a rewrite rule and $(e_2, e_2') \in R_1$ is not needed as a rewrite rule. For (2) above, let us assume that a set R of equations on E satisfies the following condition.

[Local Syntax Compatibility] For any pair (e_1, e_1') in $R \cup R^{-1}$ and any expression e_2 containing e_1 as a subexpression, let $e_2' = e_2[e_1 \leftarrow e_1']$. Then e_2' is also an expression and $e_2' \ni e_1'$. ■

Local syntax compatibility ensures that , if $e_2 \ni e_1$ and $(e_1, e_1') \in R \cup R^{-1}$ then $e_2' = e_2[e_1 \leftarrow e_1']$ is always an expression and $e_2' \ni e_1'$. Hence in the definition of R_1, R is sufficient as a set of rewrite rules ($R^{\#}$ is not needed). Notice that, even if a set R of equations satisfies local syntax compatibility, it does not necessarily follow that $R_1, R_2, \ldots$ also satisfy it. However, the latter is not required since $(\bigcup_{i\geq 1} R_i) - R$ is not needed as rewrite rules if syntactic transitivity is satisfied. To summarize the above discussion, a sufficient condition for $\tilde{R} = (\underset{R}{\rightarrow})^{\#}$ can be derived as follows.

[Definition 2.2] For a set E of expressions, define the relations $\lesssim$ and $\approx$ as follows:

(1) $e_1 \lesssim e_1'$ if and only if

for any expression e_2 in E such that $e_2 \ni e_1$,
$e_2' = e_2[e_1 \leftarrow e_1']$ is also an expression in E and $e_2' \ni e_1'$.

(2) $e \approx e'$ if and only if $e \lesssim e'$ and $e' \lesssim e$. ■

[Lemma 2.2] For a set E of expressions and a relation R on E, if the following conditions (1) and (2) are satisfied, then *operational completeness* $\bar{R} = (\underset{R}{\rightarrow})^{\#}$ holds.
(1) E satisfies syntactic transitivity.
(2) R satisfies *local syntax compatibility "e R e' implies e* $\approx$ *e' "*. ■

3. A Set of Expressions Generated by a Grammar

3.1 A Set of Expressions Generated by a Grammar

This subsection describes how a set of expressions together with a syntax relation (see 2.1) is defined by a context-free grammar. Let $G = (V_N, V_T, P)$ be a context-free grammar (cfg) without a start symbol, where V_N, V_T ($V_N \cap V_T = \phi$, ϕ denotes the empty set) and P are a finite set of non-terminal symbols, a finite set of terminal symbols and a finite set of productions, respectively. A production in P has a form $A \rightarrow \alpha$, where $A \in V_N$ and $\alpha \in (V_N \cup V_T)^+$. Let $V = V_N \cup V_T$. We write $\alpha \underset{G}{\Rightarrow} \beta$ if $\alpha = uAv$, $\beta = u\gamma v$ and $A \rightarrow \gamma \in P$ for some $u, v \in V^*$. Let $\overset{i}{\underset{G}{\Rightarrow}}$ and $\overset{*}{\underset{G}{\Rightarrow}}$ denote $(\underset{G}{\Rightarrow})^i$ and $(\underset{G}{\Rightarrow})^*$, respectively, and $\underset{G}{\Rightarrow}$, $\overset{i}{\underset{G}{\Rightarrow}}$ and $\overset{*}{\underset{G}{\Rightarrow}}$ are abbreviated as $\Rightarrow$, $\overset{i}{\Rightarrow}$ *and* $\overset{*}{\Rightarrow}$, respectively if there is no ambiguity. For each $A \in V_N$, let

$$L_G(A) = \{ e \mid A \overset{*}{\underset{G}{\Rightarrow}} e, e \in V_T^{+} \}$$

and let

$$E_G = \bigcup_{A \in V_N} L_G(A).$$

For $e \in E_G$, let

$$N_G(e) = \{ A \mid A \in V_N, A \overset{*}{\underset{G}{\Rightarrow}} e \}.$$

A cfg G specifies E_G as a set of expressions.

A *derivation tree* is defined to be a tree tr satisfying the following conditions (1) to (3). A node with at least one child is called an *internal node*.
(1) Each node in tr has a symbol in V as the label.
(2) The label of an internal node is a non-terminal symbol in V_N, and, for a tree which consists of a single node r, the label of r is a non-terminal symbol.
(3) If a node n has children $n_1, n_2, \ldots, n_k$ from left to right, and the labels of $n, n_1, n_2, \ldots, n_k$ are $X, X_1, X_2, \ldots, X_k$, respectively, then $X \rightarrow X_1X_2 \ldots X_k \in P$. ■

For a derivation tree tr, tr is called a derivation tree for α, where α is the string obtained by concatenating the labels of the leaves in tr from left to right.

If *tr* is a derivation tree for α and the label of the root of *tr* is A, then we write *tr* $: A \overset{*}{\underset{G}{\Rightarrow}} \alpha$ by using the same notation as for derivations. Similarly, if the number of the internal nodes of a derivation tree *tr* is i, then *tr* may be written as *tr* $:A \overset{i}{\underset{G}{\Rightarrow}} \alpha$. For a string $\alpha \in E_G$, a substring β of α and a derivation tree $tr : A \overset{*}{\underset{G}{\Rightarrow}} \alpha$ for α, if *tr* contains a derivation tree for β as a subtree, then β is said to be *derived from a non-terminal symbol in tr.*

A cfg G specifies the following relation $\underset{G}{\ni}$ on E_G as a syntax relation:

$e_2 \underset{G}{\ni} e_1$ if and only if

there exists a derivation tree *tr* for e_2 such that e_1 is derived from a non-terminal symbol in *tr*. ■

For a given cfg G, a set of expressions together with a syntax relation can be specified in a natural way as above by using the phrase structures defined by G. Unfortunately, however, it is undecidable whether $e \approx e'$ (see 2.3) holds in general:

[Lemma 3.1] It is undecidable for a given cfg G and expressions e and e' in E_G whether $e \approx e'$ holds.

(Proof) For a given cfg $G = (V_N, V_T, P)$ and non-terminal symbols A_1 and A_2 in V_N, we can construct another cfg G' such that $e_1 \approx e_2$ holds for two distinguished expressions e_1 and e_2 in $E_{G'}$ if and only if $L_G(A_1) = L_G(A_2)$, which is undecidable. Let $G' = (V_N', V_T', P')$, where

$$V_N' = V_N \cup \{S, H_1, H_2\} \ (V_N \cap \{S, H_1, H_2\} = \phi),$$
$$V_T' = V_T \cup \{e_1, e_2\} \ (V_T \cap \{e_1, e_2\} = \phi),$$
$$P' = P \cup \{S \to H_1 A_1, S \to H_2 A_2, H_1 \to e_1, H_2 \to e_2\}.$$

Then, it is easy to see that

$e_1 \approx e_2$ if and only if $L_G(A_1) = L_G(A_2)$. ■

3.2 A cfg with Unambiguous Structure

In general, for a cfg G and an expression $e = ue_1v$, there may be derivation trees *tr1* and *tr2* for e such that e_1 is derived from a non-terminal symbol in *tr1* while it is not the case in *tr2*.

[Definition 3.1] Let G be a cfg (V_N, V_T, P), and let *tr1* and *tr2* be derivation trees for α and β, respectively, where $\alpha, \beta \in (V_N \cup V_T)^*$ and $|\alpha| = |\beta|$. *tr1* and *tr2* are said to *have the same structure* if the following condition holds:

Let γ be a substring of α and let $\alpha = \zeta\gamma\eta$ and $\beta = \zeta'\delta\eta'$ where $|\zeta| = |\zeta'|$ and $|\eta| = |\eta'|$. γ is derived from a non-terminal symbol in *tr1* if and only if δ is derived from a non-terminal symbol in *tr2*. ■

For a cfg G, if, for each expression e in E_G, all the derivation trees for e have the same structure, then we say that G *has unambiguous structure*, or G *is with unambiguous structure*. If G has unambiguous structure and an expression e_1 is a subexpression of an expression e_2, then e_1 is always derived from a non-terminal symbol in every derivation tree for e_2.

[Lemma 3.2] It is undecidable for a given cfg G whether G has unambiguous structure.

(Proof) For a given instance I of Post's correspondence problem, a cfg G can be effectively constructed such that G has unambiguous structure if and only if I does not have any solution. ■

[Lemma 3.3] If a cfg G has unambiguous structure then E_G satisfies syntactic transitivity (see 2.3).

(Proof) Obvious by the definition of 'unambiguous structure'. ■

3.3 Global Syntax Compatibility - A Sufficient Condition for Local Syntax Compatibility -

We have already given a sufficient condition for a set R of equations to satisfy $\tilde{R} = (\xrightarrow[R]{})^{\#}$ as lemma 2.2. The condition requires that R satisfies local syntax compatibility "$R \subseteq \approx$". As will be described in 4.1, a text in ASL/* specifies a set R of equations by a set of axioms. However, it is undecidable for a given text t in ASL/* whether the set of equations specified by t satisfies local syntax compatibility (see lemma 4.1).

By the way, the sufficient condition in lemma 2.2 also requires that the set of expressions under consideration satisfies syntactic transitivity. In 4.2, we assume that a cfg G which specifies a set E_G of expressions has unambiguous structure so that E_G satisfies syntactic transitivity (we believe the assumption is a reasonable one as will be explained in 4.2). However, to the best of the authors' knowledge, even if a cfg is assumed to have unambiguous structure, it is open whether it is decidable or not for a text t whether $R \subseteq \approx$ holds, where R is the set of equations specified by t. If the problem is decidable, it becomes also decidable for given cfg's G_1, G_2 with unambiguous structures whether $E_{G_1} = E_{G_2}$. Since the class of cfl's generated by cfg's with unambiguous structures contains the class of deterministic cfl's, and the equality problem for deterministic cfl's has been one of the most famous open problems, it seems to be difficult to know whether the problem "$R \subseteq \approx$" is decidable or not.

Considering above discussions, we will define a relation, denoted by $\lozenge$, which satisfies $\lozenge \subseteq \approx$. Hence $R \subseteq \approx$ if $R \subseteq \lozenge$. Furthermore, it will be shown in 4.2 that it is decidable for a text $t = (G, AX)$ whether $R \subseteq \lozenge$ under the assumption that G has unambiguous structure. Intuitively, $e_1 \lozenge e_1'$ holds if and only if the following two conditions hold:
(1) $e_1 \approx e_1'$.
(2) For any expression e_2 containing e_1 as a subexpression, let $e_2' = e_2[e_1 \leftarrow e_1']$ (e_2' is always an expression by (1)). Then, for any derivation tree tr for e_2, there exists a derivation tree tr' for e_2' such that tr and tr' have the same structure "outside of " the subtrees for e_1 and e_1'. That is, tr and tr' have the same structure when pruning all the nodes other than the roots in the subtrees for e_1 and e_1', and all the edges in them. Similar condition holds also for any e_3 containing e_1' as a subexpression.

To define the relation $\lozenge$ on E_G more precisely, we first define the relation, denoted by $\underset{G}{\propto}$ (or $\propto$, if no ambiguity), on the powerset of the set of non-terminal symbols of G as follows.

[Definition 3.2] For a cfg $G = (V_N, V_T, P)$ and subsets N_1 and N_2 of V_N,

$N_1 \propto N_2$ if and only if

for any derivation tree

$$tr1 : A \overset{*}{\underset{G}{\Rightarrow}} uBv \ (A \in V_N, B \in N_1, u, v \in V_T^*),$$

there exists a derivation tree

$$tr2 : C \overset{*}{\underset{G}{\Rightarrow}} uDv \ (C \in V_N, D \in N_2)$$

such that $tr1$ and $tr2$ have the same structure.

[Definition 3.3] For a cfg G, define the relations $\angle$ and $\lozenge$ on E_G as follows:
(1) $e \angle e'$ if and only if $N_G(e) \propto N_G(e')$.
(2) $e \lozenge e'$ if and only if $e \angle e'$ and $e' \angle e$. ■

The property "$R \subseteq \lozenge$" is called *global syntax compatibility*. Obviously, global syntax compatibility implies local syntax compatibility.

[Lemma 3.4] It is decidable for a given cfg $G = (V_N, V_T, P)$ and subsets N_1, N_2 of V_N whether $N_1 \propto N_2$ holds.

(Proof) We can decide whether $N_1 \propto N_2$ by a technique similar to the one used in the proof of Theorem 1 and 3 in [7]. For simplicity, suppose $L_G(A) \neq \phi$ for each non-terminal symbol A in V_N. The decision algorithm stated below can be extended easily so as to be applied to a cfg which may have a non-terminal A such that $L_G(A) = \phi$. For a given cfg $G = (V_N, V_T, P)$ and subsets N_1, N_2 of V_N, execute the next procedure, PROC1. $N_1 \propto N_2$ holds if the output is "YES" and $N_1 \propto N_2$ does not hold if "NO".

[PROC1]

(Input) A cfg $G = (V_N, V_T, P)$ and subsets N_1, N_2 of V_N

(Output) "YES" or "NO"

(Step 1) Construct a cfg $G_1 = (V_N, V_T, P_1)$ from G, where

$P_1 = \{ A \to \alpha \mid A, B \in V_N, A \overset{*}{\underset{G}{\Rightarrow}} B, B \to \alpha \in P$ and $B \to \alpha$ is not a unit production (i.e., α is not a string consisting of a single non-terminal symbol) $\}$.

By the definition of G_1, $E_{G_1} = E_G$, $\underset{G_1}{\ni} = \underset{G}{\ni}$ and G_1 has no unit production.

(Step 2) Construct a cfg $G_2 = (V_{N2}, V_T, P_2)$, where

(1) V_{N2} consists of all the (newly introduced) non-terminal symbols each of which corresponds (one to one) to a non-empty subset of V_N. For each B $\in V_{N2}$, let $\sigma(B)$ denote the subset of V_N corresponding to B.

(2) $B \to u_0 B_1 u_1 \dots B_n u_n \in P_2$ $(B, B_1, \dots, B_n \in V_{N2}, u_0, u_1, \dots, u_n \in V_T^*)$,

if and only if

$\sigma(B)$ consists of exactly all the non-terminal symbols $A \in V_N$ such that there exists $A_i \in \sigma(B_i)$ $(1 \leq i \leq n)$ and $A \to u_0 A_1 u_1 \dots A_n u_n \in P_1$.

(Step 3) Construct a cfg $G_3 = (V_{N3}, V_T, P_3)$ from G_2 by removing from G_2 (1) useless non-terminal symbols (i.e., which do not generate any expression), and (2) productions which have at least one useless non-terminal symbol in left-hand side or right-hand side. Details are described in [3].

(Step 4) If the following condition holds, output "YES", and otherwise, output "NO".

(Condition) Let $m = 2^{|V_{N3}|}-1$. Let C_1 be an arbitrary non-terminal symbol of G_3 such that $A_1 \in \sigma(C_1)$ for some $A_1 \in N_1$. For any derivation tree

$tr1 : B_1 \overset{i}{\underset{G3}{\Rightarrow}} \alpha C_1 \beta \quad (B_1 \in V_{N3}, \alpha, \beta \in (V_{N3} \cup V_T)^*, i \leq m)$,

there exists a derivation tree

$tr2 : B_2 \overset{i}{\underset{G3}{\Rightarrow}} \alpha C_2 \beta \quad (B_2, C_2 \in V_{N3})$

which satisfies the following conditions (1) and (2):

(1) $A_2 \in \sigma(C_2)$ for some $A_2 \in N_2$,

(2) $tr1$ and $tr2$ have the same structure. ■

[Corollary 3.5] It is decidable for a given cfg G and expressions e, e' in E_G whether $e \angle e'$, and hence so is whether $e \lozenge e'$.

(Proof) Obvious from lemma 3.4 and the definitions of $\angle$ and $\lozenge$. ■

4. A Congruence Relation Generated by Axioms

4.1 The Algebraic Language ASL/*

ASL/* is an algebraic description language used for writing programs and/or their specifications. A pair (G, AX) is called a *text* in ASL/*, where G is a cfg without a start symbol and AX is a finite set of axioms defined below. A text $t = (G, AX)$ specifies (a) a set of expressions together with a syntactic relation by G, and (b) a set of equations (and, in turn, a congruence relation) by AX as follows:
(a) A text $t = (G, AX)$ in ASL/* specifies E_G as a set of expressions and $\underset{G}{\ni}$ as a syntax relation on E_G (see 3.1).
(b) Let V_{var} be a set of variables such that $V_{var} \cap (V_N \cup V_T) = \phi$. For each variable x in V_{var}, a non-terminal symbol in V_N, denoted by M_x, is associated with x. M_x is called *the type of* x. Let

$$G(V_{var}) = (V_N,\ V_T \cup V_{var},\ P \cup \{\ M_x \to x \mid x \in V_{var}\}).$$

An expression in $E_{G(V_{var})}$ is called an *expression with variables* (*with respect to* V_{var}).

Let $\{x_1/u_1, x_2/u_2, ..., x_n/u_n\}$ denote the substitution which substitutes strings $u_1, u_2, ..., u_n$ for variables $x_1, x_2, ..., x_n$, respectively. For a string u and a substitution θ, let $u\theta$ denote the string obtained from u by the substitution θ.

An *axiom* in ASL/* consists of
(1) variables $x_1, x_2, ..., x_n$ and their types $M_{x_1}, M_{x_2}, ..., M_{x_n}$,
and
(2) a pair (l, r) of expressions with variables (w.r.t. $\{\ x_1, x_2, ..., x_n\ \}$).
An axiom is sometimes written as

$$x_1{:}M_{x_1},\ x_2{:}M_{x_2},\ ...,\ x_n{:}M_{x_n},\quad l == r.$$

Let $t = (G, AX)$ be a text in ASL/*. Let ax be an axiom in AX which is

$$x_1{:}M_{x_1},\ x_2{:}M_{x_2},\ ...,\ x_n{:}M_{x_n},\quad l == r.$$

Define $Q_t(ax)$ as

$$Q_t(ax) = \{\ (l\theta, r\theta) \mid \theta = \{x_1/e_1, x_2/e_2, ..., x_n/e_n\} \text{ is a substitution satisfying } e_i \in L_G(M_{x_i})\ (1 \le i \le n)\ \},$$

and let

$$Q_t = \bigcup_{ax \in AX} Q_t(ax).$$

Q_t is called *the set of equations specified by the text* t. The congruence relation $\bar{Q}_t$ generated by Q_t (see 2.2) is called *the congruence relation specified by* t and denoted by $\equiv_t$. If $e \equiv_t e'$ holds for expressions e and e' in E_G, then we say that *e is congruent with e' in t.*

4.2 A Sufficient Condition for a Text to Satisfy Operational Completeness

Let $t = (G, AX)$ be a text in ASL/*. Since t specifies Q_t as a set of equations and $\equiv_t$ $(= \dot{Q}_t)$ as a congruence relation, operational completeness means $\equiv_t = (\underset{Q_t}{\rightarrow})^{\#}$. By lemma 2.2, $\equiv_t = (\underset{Q_t}{\rightarrow})^{\#}$ holds if E_G satisfies syntactic transitivity, and E_G and Q_t satisfy local syntax compatibility "$Q_t \subseteq \approx$". However, the latter is undecidable in general.

[Lemma 4.1] It is undecidable for a given text $t = (G, AX)$ in ASL/* whether E_G and Q_t satisfy local syntax compatibility "$Q_t \subseteq \approx$".

(Proof) For a given cfg G and expressions e, e' in E_G, construct a text $t = (G, AX)$, where AX consists of only one axiom $e == e'$, which contains no variables. Since $Q_t = \{ (e, e') \}$ by the definition of t, $Q_t \subseteq \approx$ if and only if $e \approx e'$, which is undecidable by lemma 3.1. ■

In the following, a sufficient condition is given for a text t in ASL/* to satisfy operational completeness $\equiv_t = (\underset{Q_t}{\rightarrow})^{\#}$ with the help of lemma 2.2. Hereafter, for a text $t = (G, AX)$, G is assumed to have unambiguous structure, which we believe is a reasonable assumption for the following reasons:

(1) If G has unambiguous structure, then E_G satisfies syntactic transitivity (lemma 3.3).

(2) As mentioned in 3.3, global syntax compatibility "$R \subseteq \Diamond$" implies local syntax compatibility "$R \subseteq \approx$". As shown in lemma 4.2, it is decidable for a text $t = (G, AX)$ to satisfy global syntax compatibility "$Q_t \subseteq \Diamond$" under the assumption that G has unambiguous structure.

(3) Even if the class of cfg's used for specifying the syntax of expressions is restricted to the class of cfg's with unambiguous structure, we can write specifications (texts) in a natural way which deal with error handling (see example 4.1 at the end of this section, where the 'popping' operation on the empty stack are considered), or sort inclusion (such as integer and real). For example, the syntax of reals (including integers) can be specified by a cfg with unambiguous structure as follows:

real	$\rightarrow$	real + real	real	$\rightarrow$	0.1
real	$\rightarrow$	int			
int	$\rightarrow$	int + int	int	$\rightarrow$	0
	$\vdots$				

On the other hand, if the class of cfg's is restricted to a smaller class than the class of cfg's with unambiguous structure, e.g., to the class of unambiguous cfg's, then the grammar for specifying the syntax of expressions in such a text as the example above becomes complicated.

[Lemma 4.2] Let $t = (G, AX)$ be a text in ASL/*, where G has unambiguous structure. It is decidable whether t satisfies global syntax compatibility "$Q_t \subseteq \Diamond$".

(Proof) Let $G = (V_N, V_T, P)$. By the definition of Q_t, global syntax compatibility is logically equivalent to

(a) for each ax in AX, $e\ Q_t(ax)\ e'$ implies $e \Diamond e'$.

Let AX^{-1} be the set of the axioms in AX, left-hand sides and right-hand sides interchanged. Then, (a) in turn is equivalent to

(b) for each ax in $AX \cup AX^{-1}$, (b1) $e\ Q_t(ax)\ e'$ implies $e \angle e'$.

Since AX is finite, it is enough to show that it is decidable for a given axiom ax whether $e\ Q_t(ax)\ e'$ implies $e \angle e'$. Let ax be an axiom in AX which is

$x_1{:}M_{x_1},\ x_2{:}M_{x_2},\ \ldots,\ x_n{:}M_{x_n},\ \ l == r.$

Then the condition (b1) holds if and only if

(c) for any substitution $\theta = \{x_1/e_1, x_2/e_2, \ldots, x_n/e_n\}$ satisfying $e_i \in L_G(M_{x_i})$ $(1 \le i \le n)$, $N_G(l\theta) \underset{G}{\approx} N_G(r\theta)$.

Let

$\Sigma = \{ (N_G(l\theta), N_G(r\theta)) \mid \theta = \{x_1/e_1, x_2/e_2, \ldots, x_n/e_n\}$ is an arbitrary substitution satisfying $e_i \in L_G(M_{x_i})\ (1 \le i \le n) \}$.

Then, the condition (c) can be paraphrased as

(c') for each pair (S_1, S_2) in Σ, $S_1 \underset{G}{\approx} S_2$.

Although the number of substitutions θ's in (c) is infinite in general, Σ is finite since it is a set of subsets of $V_N \times V_N$. Σ is effectively obtained as follows.

For each variable $x_i\ (1 \le i \le n)$, let

$$\tilde{N}_{x_i} = \{ N_G(e_i) \mid e_i \in L_G(M_{x_i}) \}$$

and for the left-hand side l of ax and sets of non-terminal symbols $N_1, N_2, \ldots, N_n$ satisfying $N_i \in \tilde{N}_{x_i}\ (1 \le i \le n)$, let

$N[l, N_1, N_2, \ldots, N_n] = \{ A \mid$ there exists $\bar{l}$ such that $A \overset{*}{\underset{G}{\Rightarrow}} \bar{l}$, where $\bar{l}$ is obtained from l by replacing each variable x_i with N_i (distinct occurrences of the same variable x_i may be replaced with distinct non-terminal symbols in N_i) $\}$

and define $N[r, N_1, N_2, \ldots, N_n]$ for the right-hand side r in the same way.
Since G has unambiguous structure, Σ coincides with

$$\{ (N[l, N_1, N_2, \ldots, N_n], N[r, N_1, N_2, \ldots, N_n]) \mid N_i \in \tilde{N}_{x_i}\ (1 \le i \le n) \}.$$

Let G_2 be a cfg obtained by executing Steps 1 and 2 of PROC1 in lemma 3.4 with G as input. Let

$$\tilde{N}_{x_i}' = \{ \sigma(B) \mid M_{x_i} \in \sigma(B) \text{ and } L_{G_2}(B) \neq \phi \}$$

for each i $(1 \leq i \leq n)$, then $\tilde{N}_{x_i} = \tilde{N}_{x_i}'$ holds for each i $(1 \leq i \leq n)$ since G has unambiguous structure. Hence Σ can be effectively obtained. To summarize above arguments, we can decide whether $Q_t \subseteq \lozenge$ as follows: Execute the following procedure PROC2 with a given text $t = (G, AX)$ as input. If the answer is "YES" then $Q_t \subseteq \lozenge$, and otherwise, $Q_t \subseteq \lozenge$ does not hold.
[PROC2]
(Input) A text $t = (G, AX)$ in ASL/*
(Output) "YES" or "NO"
(Step 1) Execute Steps 1 and 2 of PROC1 in the proof of lemma 3.4 with G as input. Let G_2 be a cfg constructed in Step 2 of PROC1.
(Step 2) For each axiom ax in $AX \cup AX^{-1}$, execute Steps 3 and 4 below. If the answer is always "YES", then output "YES", and otherwise, output "NO".
(Step 3) Let a given axiom be

$$x_1{:}M_{x_1},\ x_2{:}M_{x_2},\ \dots,\ x_n{:}M_{x_n},\ \ l == r.$$

Compute $\tilde{N}_{x_i}$ for each i $(1 \leq i \leq n)$, where

$$\tilde{N}_{x_i} = \{\ \sigma(B)\ \mid\ M_{x_i} \in \sigma(B) \text{ and } L_{G_2}(B) \neq \phi\}.$$

(Step 4) For each $N_i \in \tilde{N}_{x_i}$ $(1 \leq i \leq n)$, determine whether

$$N[l, N_1, N_2, \dots, N_n] \underset{G}{\cong} N[r, N_1, N_2, \dots, N_n] \qquad (*)$$

which is decidable by lemma 3.4. If (*) holds for every pair $N_1, N_2, \dots, N_n$, then output "YES", and otherwise, output "NO". ■

Now, the main theorem can be derived as a corollary of lemma 2.2, lemma 3.3 and lemma 4.2.

[Theorem 4.3] Let $t = (G, AX)$ be a text in ASL/*, where G has unambiguous structure. It is decidable whether t satisfies global syntax compatibility $Q_t \subseteq \lozenge$ which implies operational completeness $\equiv_t = (\underset{Q_t}{\rightarrow})^{\#}$. ■

[Example 4.1] Figure 1 is a text $t = (G, AX)$ in ASL/* which defines a data type, stack of non-negative integers, such as:

> Popping the empty stack results in the error state (denoted by *ERRSTACK*) and no operation is defined on *ERRSTACK*. If one tries to know the top of the empty stack, the distinguished value, *ERRINT*, is returned.

Lines (1) to (14) define the productions of G. The symbols appearing in the left-hand side of some production are non-terminal symbols, and the other symbols appearing in the right-hand sides of the productions are terminal symbols. Lines (17) to (22) define the axioms in AX. Line (15) and (16) defines that the type of variables s and i are *stack* and *int*, respectively.

```
(1)   stack        → NEW_STACK
(2)   stack        → PUSH(stack, int)
(3)   stack&err    → stack
(4)   stack&err    → ERRSTACK
(5)   stack&err    → PUSH(stack&err, int)
(6)   stack&err    → POP(stack&err)
(7)   int          → 0
(8)   int          → SUCC(int)
(9)   int&err      → int
(10)  int&err      →ERRINT
(11)  int&err      → TOP(stack&err)
(12)  bool         → TRUE
(13)  bool         → FALSE
(14)  bool         → ISNEW(stack&err)
(15)  stack  s
(16)  int    i
(17)  ISNEW(NEW_STACK)    == TRUE
(18)  ISNEW(PUSH(s, i))   == FALSE
(19)  TOP(NEW_STACK)      == ERRINT
(20)  TOP(PUSH(s, i))     == i
(21)  POP(NEW_STACK)      == ERRSTACK
(22)  POP(PUSH(s, i))     == s
```

Figure 1 A Specification of Stack with Error State

Non-terminal symbol *int* generates all the non-negative integers ; $L_G(int) = \{ SUCC^n(0) \mid n \geq 0 \}$. Let n $(n \geq 0)$ denote $SUCC^n(0)$ for notational simplicity. *int&err* generates all the non-negative integers together with the distinguished value *ERRINT*, i.e., $L_G(int\&err) = L_G(int) \cup \{ERRINT\}$. Similarly, *stack* generates all the expressions each of which denotes a legal states of the stack and $L_G(stack\&err) = L_G(stack) \cup \{ERRSTACK\}$.

Operations *ISNEW, POP*, and *TOP* are defined only on the expressions in $L_G(stack)$. For example,

TOP(PUSH(NEW_STACK, 3))

is congruent with *3* by the axiom (20) while

TOP(PUSH(POP(NEW_STACK), 3)) (*)

is not congruent with any n $(n \geq 0)$. That is, the value of (*) is not defined, which means that no operation is defined once the stack goes into error state.

Let us execute PROC2 with the text shown in Figure 1 as input. By executing Steps 1 to 3, we know

$\tilde{N}_s = \{\{stack, stack\&err\}\}$

$\tilde{N}_i = \{\{int, int\&err\}\}$

If we execute Step 4 for the axiom (18), only the pair $S_1 = \{bool\}$ and $S_2 = \{bool\}$ is obtained and both $S_1 \propto S_2$ and $S_2 \propto S_1$ hold obviously. Similar arguments hold for (17), (19) and (21). For (20), the pair $S_1 = \{int\&err\}$ and $S_2 = \{int, int\&err\}$ is obtained and $S_1 \propto S_2$ holds while $S_2 \propto S_1$ does not hold. The reason why the latter does not hold is that *PUSH* can take as its second argument only expressions in $L_G(int)$ (and so is the first argument of *SUCC*). For example, if in the expression *PUSH(NEW_STACK, 3)*, the subexpression *3* is replaced with the congruent expression *TOP(PUSH(NEW_STACK, 3))*, the resulting string

PUSH(NEW_STACK, TOP(PUSH(NEW_STACK, 3)))

is not an expression. The reason is that *3* is in $L_G(int)$ while *TOP(PUSH(NEW_STACK, 3))* is not. For (22), we obtain the pair $S_1 = \{stack\&err\}$ and $S_2 = \{stack, stack\&err\}$ and in this case both $S_1 \propto S_2$ and $S_2 \propto S_1$ hold. The reason why the latter holds is that every operation which can take an expression in $L_G(stack)$ as its argument can also take an expression in $L_G(stack\&err)$ as its argument. ■

The text in Figure 1 does not satisfy global syntax compatibility since $\{int, int\&err\} \underset{G}{\propto} \{int\&err\}$ does not hold. For a text $t = (G, AX)$ which does not satisfy global syntax compatibility, by adding certain productions to the set of productions of G, t may be transformed into a text $t' = (G', AX)$ such that

(#1) t' satisfies global syntax compatibility, and

(#2) for any expressions e and e' in E_G, $e \equiv_t e'$ if and only if $e \equiv_{t'} e'$.

Let $t' = (G', AX)$, where AX is the same as in Figure 1 and G' is obtained from G by adding the following productions to the set of productions of G:

$$
\begin{aligned}
&\mathit{stack} &&\rightarrow \mathit{PUSH}(\mathit{stack}, \mathit{int\&err}) \\
&\mathit{stack\&err} &&\rightarrow \mathit{PUSH}(\mathit{stack\&err}, \mathit{int\&err}) \\
&\mathit{int\&err} &&\rightarrow \mathit{SUCC}(\mathit{int\&err})
\end{aligned}
$$

Then, $\{int, int\&err\} \underset{G'}{\propto} \{int\&err\}$ holds, and the conditions (#1) and (#2) are satisfied although E_G is properly contained in $E_{G'}$, e.g., *PUSH(NEW_STACK, ERRINT)* is in $E_{G'}$ while E_G does not contain it.

In general, if a text $t = (G, AX)$ is transformed into a text $t' = (G', AX)$ by adding certain productions, $E_G \subseteq E_{G'}$ and $\underset{G}{\ni} \subseteq \underset{G'}{\ni}$ always hold but $E_{G'} \subseteq E_G$ and $\underset{G'}{\ni} \subseteq \underset{G}{\ni}$ are not always true. It is under study to investigate a transformation from a given text into a text which satisfies the conditions (#1) and (#2) above.

5. Conclusion

In this paper, a congruence relation is extended on a set of expressions generated by a cfg so that the semantics of specifications or programs which deal with error handling and/or sort inclusion can be defined in a simple algebraic framework. Next, a sufficient condition is shown under which the operational semantics of the congruence relation generated by a set of axioms can be defined simply in the sense that operational completeness holds. Although only unconditional axioms are considered in this paper, conditional axioms can also be used in ASL/*. A conditional axiom has a form such as

$$x_1{:}M_{x_1}, x_2{:}M_{x_2}, \ldots, x_n{:}M_{x_n}, \ l_1 == r_1, l_2 == r_2, \ldots, l_m == r_m ==> l == r$$

which means "if $l_1, l_2, \ldots,$ and l_m are congruent with $r_1, r_2, \ldots,$and r_m, respectively, then l must be congruent with r." The congruence relation generated by a text which contains conditional axioms can be defined as simply as in the unconditional case [6]. Intuitively, the operational semantics of the above axiom is that, if $r_1, r_2, \ldots,$ and r_m are obtained by rewriting from $l_1, l_2, \ldots,$ and l_m, respectively, then l can be rewritten as r. It is an open problem to find a sufficient condition under which the operational semantics of the congruence relation generated by a text which contains conditional axioms can be defined simply.

Acknowledgments The authors would like to thank Associate Professor Yuji SUGIYAMA of Department of Information and Computer Sciences, Osaka University, for his fruitful discussions.

References

[1] J. A. Goguen, J. W. Thatcher and E. G. Wagner: *"An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types"*, IBM Research Report, RC 6487 (1976), also in R. Yeh (ed.): *'Current Trends in Programming Methodology IV: Data Structuring'*, Prentice Hall, pp.80-144 (1978).

[2] J. A. Goguen: *"Abstract Errors for Abstract Data Types"*, Proc. of IFIP Working Conf. on Formal Description of Programming Concepts, pp.21.1-21.32 (1977).

[3] J. E. Hopcroft and J. D. Ullman: *"Introduction to Automata Theory, Languages and Computation"*, Addison-Wesley (1979).

[4] G. Huet and D. C. Oppen: *"Equations and Rewrite Rules: A Survey"*, in R. V. Book (ed.): *'Formal Language Theory: Perspectives and Open Problems'*, pp.349-393, Academic Press (1980).

[5] K. Inoue, H. Seki, K. Taniguchi and T. Kasami: *"Compiling and Optimizing Methods for the Functional Language ASL/F"*, Science of Computer Programming, Vol.7, No.3, pp.297-312 (1986).

[6] T. Kasami, K. Taniguchi, Y. Sugiyama and H. Seki: *"Principles of Algebraic Language ASL/*"*, Trans. IECE Japan, Vol.J69-D, No.7, pp.1066-1074 (July 1986) (in Japanese).

[7] R. McNaughton: *"Parenthesis Grammars"*, J. ACM, Vol.14, No.3, 490-500 (July 1967).

[8] H. Seki, K. Inoue, Y. Sugiyama, K. Taniguchi and T. Kasami: *"Design Principles of Algebraic Specification Language ASL/*"*, Paper of Technical Group, IECE Japan, AL84-55 (Jan. 1985) (in Japanese).