Assignment of Data Types to Words in a Natural Language Specification

Yasunori ISHIHARA†, Atsushi OHSAKI††, Hiroyuki SEKI†, and Tadao KASAMI†

† Graduate School of Information Science
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-01 JAPAN
E-mail: ishihara@is.aist-nara.ac.jp

†† Sumitomo Metal Industries
Osaka-shi 541 JAPAN

ABSTRACT

When a natural language specification is translated into a formal one, it is important for objects and operations appearing in the natural language specification to be appropriately classified according to the framework of data types in the formal specification. In this paper, we propose a semi-automatic method of constructing a context-free grammar (cfg) representing an assignment of data types to words in a given natural language specification. In our method, a cfg is mechanically constructed from sample sentences in a natural language specification, where the cfg represents type declarations of expressions and type hierarchy. Then, the cfg is appropriately modified by adding nonterminals/production rules that represent type inclusion relations. In this modification process, candidates for the productions to be added are presented to the user. Finally, the cfg is simplified based on structural equivalence. The result of applying this method to a part of the OSI session protocol specification (39 sentences) is also presented. There was an example in which ambiguity of anaphoric bindings was solved by type checking based on the resulting cfg.

1. Introduction

In a software development process, informal requirements and/or specifications are often written in a natural language since they become readable and the intuitive meanings understandable. Such an informal specification is (manually or semi-automatically) translated into a formal specification so that ambiguity, incompleteness, and inconsistency of the informal specification are reduced. Systematic refinement and verification methods can be applied to the translated specification.

Most of formal specification languages have a concept of (abstract) data types. When an informal specification is translated into a formal one, it is important to appropriately classify objects and operations appearing in the informal specification by the following reasons:

- The formal specification becomes simple and concise since some of semantic descriptions (axioms) can be shifted to, and encapsulated in type declarations. Hence, the refinement step will be easily done; and
- Ambiguity or inconsistency of the informal specification can be detected or reduced by means of type checking.

However, as the size of an informal specification grows, it becomes time-consuming to classify them manually.

This paper proposes a method of assigning data types to objects and operations appearing in a given natural language specification. As a formal specification language, algebraic specification language ASL [4] is adopted. An algebraic specification in ASL is a pair $SPEC = (G, AX)$ of a context-free grammar (cfg) G and a set AX of axioms. G specifies the set of expressions and their syntax, and AX specifies their semantics. One of the reasons why this paper adopts ASL is that data types are simply defined by a cfg G in ASL. Subtype relation can also be represented as the derivation relation in G. Hence our goal is to derive an appropriate cfg G, i.e., to define the syntax of expressions, from a given natural language specification. Another reason is that we have already proposed a method of translating a natural language specification $SPEC_{\mathrm{NL}}$ into an algebraic specification $SPEC_{\mathrm{LF}} = (G_{\mathrm{LF}}, AX_{\mathrm{LF}})$ whose axioms are in the form of logical formulas in Refs. [2] and [9]. This translation method assumes that the part of G_{LF} assigning data types to words in $SPEC_{\mathrm{NL}}$ is given as lexical items.

Example 1: Consider the following sentence in the OSI session protocol specification [3]:

> A valid incoming MAJOR SYNC POINT SPDU results in an S-SYNC MAJOR indication.

Table 1 Algebraic specification derived from the natural language sentence in Example 1.

- Productions:

$$\begin{array}{rcl} \mathsf{Bool} & \rightarrow & \mathsf{valid(In\_SPDU)} \\ \mathsf{Bool} & \rightarrow & \mathsf{incoming(In\_SPDU)} \\ \mathsf{Bool} & \rightarrow & \mathsf{MAP(SPDU)} \\ \mathsf{Bool} & \rightarrow & \mathsf{SSYNMind(SSprm)} \\ \mathsf{Bool} & \rightarrow & \mathsf{result\_in(In\_event, Out\_event)} \\ & \vdots & \end{array}$$

- Axioms:

$$\bar{x}_1 : \mathsf{In\_SPDU} \quad \bar{x}_2 : \mathsf{Out\_SSprm}$$
$$\forall \bar{x}_1 \exists \bar{x}_2 (\mathsf{valid}(\bar{x}_1) \wedge \mathsf{incoming}(\bar{x}_1) \wedge \mathsf{MAP}(\bar{x}_1) \supset$$
$$(\mathsf{SSYNMind}(\bar{x}_2) \wedge \mathsf{result\_in}(\bar{x}_1, \bar{x}_2))) \ == \ \mathsf{true}$$

This sentence is translated into the algebraic specification shown in Table 1, where "implicitly specified parameters" introduced in Ref. [2] are omitted for simplicity. The first production specifies that adjective "valid" can modify only the noun phrases to which type $\mathsf{In\_SPDU}$ (denoting "incoming session protocol data units") is assigned. And, in the axiom, $\bar{x}_1$ is a variable of type $\mathsf{In\_SPDU}$ corresponding to "MAJOR SYNC POINT SPDU." In the translation method proposed in Refs. [2] and [9], such type information is assumed to be given. □

By combining the translation method with the assignment method proposed in this paper, the whole translation procedure becomes more refined and automated.

Assume that a grammar G_{NL} of a natural language (English in this paper) is given. We can define a mapping $\theta_{G_{\mathrm{NL}}}$ from natural language sentences into algebraic expressions such that, roughly speaking, each verb corresponds to a function symbol and each noun corresponds to a variable of axioms. Let $SPEC_{\mathrm{NL}}$ be a specification written in the natural language, and let *EXP* be the set of all the subexpressions in $\{\theta_{G_{\mathrm{NL}}}[S] \mid S$ is a sentence appearing in *SPEC*$_{\mathrm{NL}}\}$. Our method of constructing G_{LF} for *EXP* consists of the following three phases (see Fig. 1):

(1) A set of nonterminals and a set of production rules to generate *EXP* are mechanically constructed. Then subtype relation to be satisfied by G_{LF} is generated, based on the function-argument relationship (refer to Sect. 3) in each expression in *EXP*;

(2) A human translator modifies G_{LF} so that it generates $\theta_{G_{\mathrm{NL}}}[S']$ for each $S' \in \mathcal{S}'$, where $\mathcal{S}'$ is a set of some semantically correct sentences not in *SPEC*$_{\mathrm{NL}}$, in the following way:

   - she/he groups some nonterminals (i.e., data types) together and introduces a new supertype of them, and
   - she/he augments the subtype relation to be satisfied by G_{LF};

   and

(3) G_{LF} is constructed from the set of nonterminals, the set of productions, and the subtype relation. G_{LF} is simplified based on structural equivalence [10] of cfg's.

Phases (2)–(3) are repeated until G_{LF} becomes appropriate.

Based on the proposed assignment method, the authors implemented a prototype system. By using this system, data types were assigned to words in a part of the OSI session protocol specification (39 sentences). There was an example in which ambiguity of anaphoric bindings was solved by type checking based on the resulting assignment (i.e., cfg).

This paper is organized as follows. Section 2 summarizes ASL. Sections 3–5 explain the phases (1)–(3) above, respectively. Section 6 shows the example in which ambiguity of anaphoric bindings is solved by a cfg obtained by our method. Section 7 describes an assignment system implemented according to the proposed method. Section 8 summarizes this paper.

2. Algebraic Specification Language ASL

As stated in Sect. 1, this paper adopts ASL [4] as an algebraic specification language. A specification in ASL is a pair *SPEC* = (G, AX) of a cfg G and a set AX of axioms.

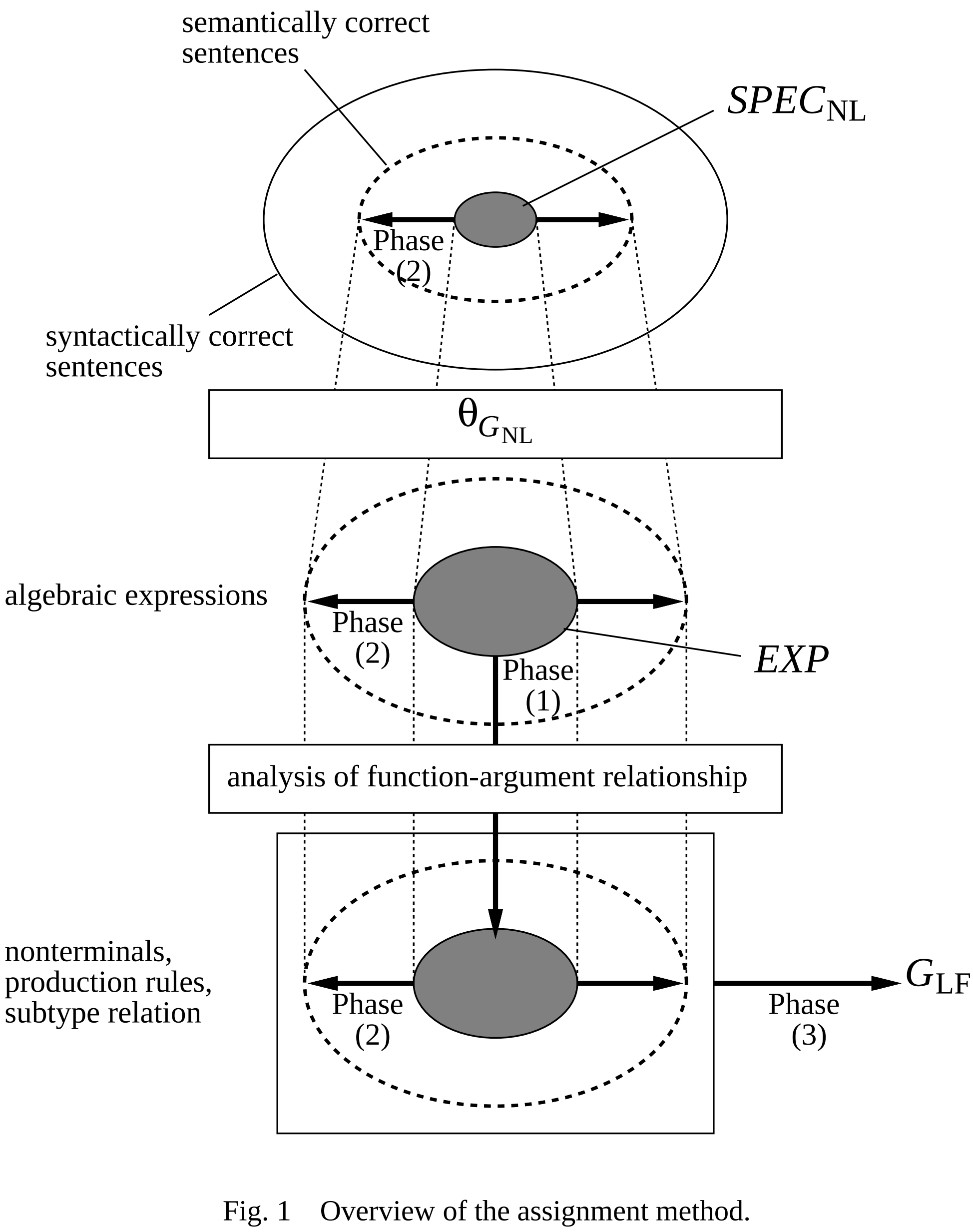

Fig. 1 Overview of the assignment method.

Let $G = (N, T, P)$, where N, T, and P are sets of nonterminals, terminals, and productions respectively. Let $A \underset{G}{\Rightarrow} \alpha$ denote that $A \in N$ generates $\alpha \in (N \cup T)^*$ by one-step derivation in G. Let $\overset{*}{\underset{G}{\Rightarrow}}$ be the reflexive and transitive closure of $\underset{G}{\Rightarrow}$. Moreover, let $L_G[A] = \{w \in T^* \mid A \overset{*}{\underset{G}{\Rightarrow}} w\}$, and let $L_G = \bigcup_{A \in N} L_G[A]$. An element in L_G is called an expression (in the specification *SPEC*). N corresponds to the set of data types (sorts); A nonterminal A is sometimes called "data type A" and an expression in $L_G[A]$ may be called "an expression of type A." For $A, A' \in N$, we say that A is a subtype of A' (or A' is a supertype of A) if $A' \overset{*}{\underset{G}{\Rightarrow}} A$. T corresponds to the set of function symbols. P corresponds to the set of signatures of functions and definitions of subtype relation. For example, consider the following productions:

$$\mathsf{Real} \rightarrow 0.1, \tag{1}$$

$$\mathsf{Real} \rightarrow \mathsf{Int}, \tag{2}$$

$$\mathsf{Int} \rightarrow 0, \tag{3}$$

$$\mathsf{Int} \rightarrow \mathsf{succ}(\mathsf{Int}), \tag{4}$$

$$\mathsf{Bool} \rightarrow f(\mathsf{Real}). \tag{5}$$

Production (2) specifies that Int is a subtype of Real. Production (4) declares that succ takes an expression of type Int as an argument and its return value is of type Int. Production (5) declares that f takes an expression of type Real as an argument and its return value is of type Bool.

An axiom is a pair $e == e'$ of expressions with variables. A variable of an axiom is denoted by a symbol with the upper bar (e.g., $\bar{x}$). With each variable $\bar{x}$ in an axiom a nonterminal $A_{\bar{x}}$ is associated (declared by "$\bar{x} : A_{\bar{x}}$" in the specification), and an arbitrary expression in $L_G[A_{\bar{x}}]$ can be substituted into $\bar{x}$. The least congruence relation that satisfies all the axioms in AX is denoted by $\equiv_{SPEC}$. See Ref. [4] for details.

In this paper, a fixed specification $SPEC_0 = (G_0, AX_0)$ of primitive data types (e.g., integer, Boolean, set, and so on) is assumed. Let Bool be a nonterminal which generates Boolean expressions.

3. Construction of a Grammar

Table 2 shows six sentences containing phrase "results in" in Ref. [3]. In the rest of this paper, these sentences are used for explaining our assignment method.

We suppose that a grammar G_{NL} of a natural language is given. In Refs. [2] and [9], a mapping $\theta_{G_{\mathrm{NL}}}$ from natural language sentences into algebraic expressions is defined. By $\theta_{G_{\mathrm{NL}}}$, each noun with no arguments is mapped into a variable and a prefix predicate symbol, called the restriction of the noun, which takes the variable as its argument. For example, in the axiom in Table 1, $\mathsf{MAP}(\bar{x}_1)$ is the restriction of "MAJOR SYNC POINT SPDU" denoting that "$\bar{x}_1$ is a MAJOR SYNC POINT SPDU." On the other hand, each noun with arguments (e.g., gerund) is mapped into a prefix function symbol. Each verb or modifier is mapped into a prefix predicate symbol. Let $\mathit{SPEC}_{\mathrm{NL}}$ be a natural language specification, and let *EXP* be the set of all the subexpressions in $\{\theta_{G_{\mathrm{NL}}}[S] \mid S$ is a sentence appearing in $\mathit{SPEC}_{\mathrm{NL}}\}$.

Example 2: From sentence (S6) in Table 2, which is the same sentence as in Example 1, the following set of expressions is obtained:

$$\{\mathsf{valid}(\bar{x}_1), \mathsf{incoming}(\bar{x}_1), \mathsf{MAP}(\bar{x}_1),$$
$$\mathsf{SSYNMind}(\bar{x}_2), \mathsf{result\_in}(\bar{x}_1, \bar{x}_2), \bar{x}_1, \bar{x}_2\}.$$

□

In the first phase of our method, (1) a cfg $G = (N, T, P)$ which generates *EXP* and (2) a subtype relation R to be satisfied by G_{LF} are mechanically constructed as shown in the following. G_{LF} is obtained by modifying G in the second and third phases.

The initial values of N, T, and P are the following sets:

$$\begin{aligned} T &= \{\text{","}, \text{"("}, \text{")"}\} \\ &\quad \cup \{f \mid f \text{ is a function (predicate) symbol appearing in } \mathit{EXP}\}, \\ N &= \{A_f \mid f \in T \text{ is a function (predicate) symbol}\} \\ &\quad \cup \{A_{(f,1)}, \ldots, A_{(f,n)} \mid \\ &\qquad f \in T \text{ is an } n\text{-ary function (predicate) symbol}\} \end{aligned}$$

Table 2 Sentences containing "results in" in Ref. [3].

(S1) An S-CONNECT request results in the assignment of a transport connection.

(S2) An S-CONNECT accept response results in an ACCEPT SPDU.

(S3) A valid incoming ACCEPT SPDU results in an S-CONNECT accept confirm.

(S4) A valid incoming ABORT SPDU results in sending an ABORT ACCEPT SPDU.

(S5) An S-SYNC MAJOR request results in a MAJOR SYNC POINT SPDU.

(S6) A valid incoming MAJOR SYNC POINT SPDU results in an S-SYNC MAJOR indication.

$$
\begin{aligned}
& \quad \cup \{A_{\bar{x}} \mid \bar{x} \text{ is a variable appearing in } \mathit{EXP}\}, \\
P \;=\; & \{A_f \rightarrow f(A_{(f,1)}, \ldots, A_{(f,n)}) \mid \\
& \qquad f \in T \text{ is an } n\text{-ary function (predicate) symbol}\} \\
& \cup\{A_c \rightarrow c \mid c \in T \text{ is a 0-ary function symbol}\}.
\end{aligned}
$$

The intuitive meaning of each nonterminal is as follows: A_f denotes the type of the return value of f. $A_{(f,i)}$ denotes the type of the i-th argument of f. $A_{\bar{x}}$ denotes the type of $\bar{x}$.

Example 3: From the six sentences shown in Table 2, the production rules shown in Table 3 are constructed. In the table, A_f is denoted by `A[f]`, and $A_{(f,i)}$ is denoted by `A[f,i]`. □

Next, a subtype relation R to be satisfied by G_{LF} is obtained as follows:

- For each variable $\bar{x}$ and function symbol f in *EXP*, if $f(\ldots, \underset{i}{\bar{x}}, \ldots)$ appears in *EXP*, then $A_{(f,i)} \overset{*}{\Rightarrow} A_{\bar{x}}$ is in R. Intuitively, the type of $\bar{x}$ must be a subtype of that of the i-th argument of f.

- For each function symbols g and f in *EXP*, if $f(\ldots, \underset{i}{g(\cdots)}, \ldots)$ appears in *EXP*, then $A_{(f,i)} \overset{*}{\Rightarrow} A_g$ is in R. Intuitively, the type of the return value of g must be a subtype of that of the i-th argument of f.

Table 3 Production rules constructed from the sentences in Table 2.

```
A[abort_accept_spdu] ---> abort_accept_spdu(A[abort_accept_spdu,1])
A[abort_spdu] ---> abort_spdu(A[abort_spdu,1])
A[accept_spdu] ---> accept_spdu(A[accept_spdu,1])
A[assignment] ---> assignment(A[assignment,1])
A[incoming] ---> incoming(A[incoming,1])
A[major_sync_point_spdu]
    ---> major_sync_point_spdu(A[major_sync_point_spdu,1])
A[results_in] ---> results_in(A[results_in,1],A[results_in,2])
A[s_connect_accept_confirm]
    ---> s_connect_accept_confirm(A[s_connect_accept_confirm,1])
A[s_connect_accept_response]
    ---> s_connect_accept_response(A[s_connect_accept_response,1])
A[s_connect_request]
    ---> s_connect_request(A[s_connect_request,1])
A[s_sync_major_indication]
    ---> s_sync_major_indication(A[s_sync_major_indication,1])
A[s_sync_major_request]
    ---> s_sync_major_request(A[s_sync_major_request,1])
A[sending] ---> sending(A[sending,1],A[sending,2])
A[transport_connection]
    ---> transport_connection(A[transport_connection,1])
A[valid] ---> valid(A[valid,1])
```

Example 4: From the six sentences shown in Table 2, the subtype relation shown in Table 4 is constructed. The phrase to which each variable appearing in Table 4 corresponds is shown in Table 5. □

4. Augmentation of the Grammar

In the second phase, the set N of nonterminals and the subtype relation R are augmented appropriately by a human translator. This augmentation method consists of the following three steps (a)–(c):

Step (a): According to the semantics of a natural language adopted in Refs. [2] and [9], the data type A of the return value of each function corresponding to a verb or a modifier, or the restriction of each noun must be Bool. For such a data type A, $\mathsf{Bool} \overset{*}{\Rightarrow} A$ and $A \overset{*}{\Rightarrow} \mathsf{Bool}$ are added to R.

Example 5: To the subtype relation obtained in the previous section, the subtype relation shown in Table 6 is added, where "`A <==> B`" means "$\mathtt{A} \overset{*}{\Rightarrow} \mathtt{B}$ and $\mathtt{B} \overset{*}{\Rightarrow} \mathtt{A}$." □

Step (b): A natural language specification often defines and uses some new noun phrases (such as "S-CONNECT request" in Ref. [3]). Such definitions of new noun phrases should be translated into a part of cfg as well as axioms.

Let $w_1, \ldots, w_n$ be new noun phrases which have a common property (such as session service primitives or SPDUs illustrated in Example 6 below). Let $\bar{x}_{i1}, \ldots, \bar{x}_{im_i}$ $(1 \leq i \leq n)$ be variables corresponding to w_i (there are more than one such variables in general since a variable is introduced for each occurrence of w_i). Then, a data type, say A, denoting the common property is introduced and added into N, and, for each $\bar{x}_{ij}$ $(1 \leq i \leq n$, $1 \leq j \leq m_i)$, subtype relation $A \overset{*}{\Rightarrow} A_{\bar{x}_{ij}}$ is added to R.

Example 6: In Ref. [3], there is a description of SS primitives and SPDUs as follows:

> Information is transferred to and from the SS-user using the session service primitives listed in table 1. Table 1 also defines the SPDUs associated with each of the service primitives.

Table 4 Subtype relation constructed from the sentences in Table 2.

```
A[abort_accept_spdu,1] ===> A[_x7]
A[abort_spdu,1] ===> A[_x9]
A[accept_spdu,1] ===> A[_x3]
A[accept_spdu,1] ===> A[_x6]
A[assignment,1] ===> A[_x1]
A[incoming,1] ===> A[_x6]
A[incoming,1] ===> A[_x9]
A[incoming,1] ===> A[_x13]
A[major_sync_point_spdu,1] ===> A[_x10]
A[major_sync_point_spdu,1] ===> A[_x13]
A[results_in,1] ===> A[_x2]
A[results_in,1] ===> A[_x4]
A[results_in,1] ===> A[_x6]
A[results_in,1] ===> A[_x9]
A[results_in,1] ===> A[_x11]
A[results_in,1] ===> A[_x13]
A[results_in,2] ===> A[_x3]
A[results_in,2] ===> A[_x5]
A[results_in,2] ===> A[_x10]
A[results_in,2] ===> A[_x12]
A[results_in,2] ===> A[assignment]
A[results_in,2] ===> A[sending]
A[s_connect_accept_confirm,1] ===> A[_x5]
A[s_connect_accept_response,1] ===> A[_x4]
A[s_connect_request,1] ===> A[_x2]
A[s_sync_major_indication,1] ===> A[_x12]
A[s_sync_major_request,1] ===> A[_x11]
A[sending,1] ===> A[_x8]
A[sending,2] ===> A[_x7]
A[transport_connection,1] ===> A[_x1]
A[valid,1] ===> A[_x6]
A[valid,1] ===> A[_x9]
A[valid,1] ===> A[_x13]
```

Table 5   Variables appearing in Table 4.

Variable	Sentence	Phrase
`A[_x1]`	(S1)	“a transport connection”
`A[_x2]`	(S1)	“an S-CONNECT request”
`A[_x3]`	(S2)	“an ACCEPT SPDU”
`A[_x4]`	(S2)	“an S-CONNECT accept response”
`A[_x5]`	(S3)	“an S-CONNECT accept confirm”
`A[_x6]`	(S3)	“a (valid incoming) ACCEPT SPDU”
`A[_x7]`	(S4)	“an ABORT ACCEPT SPDU”
`A[_x8]`	(S4)	the subject of “sending”
`A[_x9]`	(S4)	“a (valid incoming) ABORT SPDU”
`A[_x10]`	(S5)	“a MAJOR SYNC POINT SPDU”
`A[_x11]`	(S5)	“an S-SYNC MAJOR request”
`A[_x12]`	(S6)	“an S-SYNC MAJOR indication”
`A[_x13]`	(S6)	“a (valid incoming) MAJOR SYNC POINT SPDU”

Table 6 Subtype relation added at Step (a).

```
Bool <==> A[abort_accept_spdu]
Bool <==> A[abort_spdu]
Bool <==> A[accept_spdu]
Bool <==> A[incoming]
Bool <==> A[major_sync_point_spdu]
Bool <==> A[results_in]
Bool <==> A[s_connect_accept_confirm]
Bool <==> A[s_connect_accept_response]
Bool <==> A[s_connect_request]
Bool <==> A[s_sync_major_indication]
Bool <==> A[s_sync_major_request]
Bool <==> A[transport_connection]
Bool <==> A[valid]
```

Table 1 of Ref. [3], whose caption is "session service primitives," has three columns: The first one is "service," the second one is "primitives," and the third one is "associated SPDUs." According to the table, the subtype relation shown in Table 7 is added, where "`A ===> B1|...|Bn`" means "$\mathtt{A} \stackrel{*}{\Rightarrow} \mathtt{B1}, \ldots, \mathtt{A} \stackrel{*}{\Rightarrow} \mathtt{Bn}$." In this example, a supertype `SPDU` of the types of the variables corresponding to "ABORT ACCEPT SPDU," "ABORT SPDU," or "MAJOR SYNC POINT SPDU" is introduced. Similarly, for noun phrases denoting SS primitives, a supertype `SSprm` is introduced.

In the case of Ref. [3], Step (b) can be semi-automated by considering the meanings of verbs such as "list" and "define," and fixing the semantics of tables. For example, consider sentence "Xs are listed in table Y." Suppose that noun phrases $w_1, \ldots, w_n$ are listed in table Y. Let $\bar{x}_{i1}, \ldots, \bar{x}_{im_i}$ ($1 \leq i \leq n$) be variables corresponding to w_i. Then, the sentence and table are translated into a data type A_X and subtype relation $A_X \stackrel{*}{\Rightarrow} A_{\bar{x}_{ij}}$ ($1 \leq i \leq n$, $1 \leq j \leq m_i$). □

Step (c): To make R appropriate, natural language phrases which are semantically correct

Table 7   Subtype relation added at Step (b).

```
SPDU ===> A[_x3]|A[_x6]|A[_x7]|A[_x9]|A[_x10]|A[_x13]
SSprm ===> A[_x2]|A[_x4]|A[_x5]|A[_x11]|A[_x12]
```

but not in $SPEC_{\mathrm{NL}}$ are to be taken into account. The syntax of the restrictions of nouns is also to be considered. However, this is time-consuming when the size of the input natural language specification is large.

We use the following heuristics:

Heuristics 1: Let A and B be nonterminals. Suppose that for any nonterminal C, $B \stackrel{*}{\Rightarrow} C$ whenever $A \stackrel{*}{\Rightarrow} C$. Then, A may be a subtype of B. □

Heuristics 2: Let A and B be nonterminals. Suppose that for any nonterminal C, $C \stackrel{*}{\Rightarrow} A$ if and only if $C \stackrel{*}{\Rightarrow} B$. Then, $A \stackrel{*}{\Rightarrow} B$ or $B \stackrel{*}{\Rightarrow} A$ or both of them may hold. □

All the tuples of nonterminals satisfying the conditions in the above heuristics can be mechanically generated. A human translator examines each tuple by considering the meanings of words which have type A or B, and decides whether the tuple should be added to R.

Example 7: Suppose that a human translator decides that the data types of the arguments of the restrictions on SPDUs are the same, and she/he also decides similarly for SS primitives. Then, the subtype relation shown in Table 8 is added to R.

Next, according to Heuristics 1, the set of candidates shown in Table 9 is obtained. `nt1` and `nt2` are the equivalence classes of $\stackrel{*}{\Leftrightarrow}$, and the elements in these equivalence classes are also shown in Table 9. A human translator examines each candidate. For example, for the first candidate in Table 9, she/he examines whether for any natural language word or phrase w, if "incoming w" is semantically correct phrase, then the type of w is `SPDU`. For the eighth candidate, she/he decides whether `abort_accept_spdu`, `abort_spdu`, etc. take expressions of type `SPDU` as their first argument. According to such examination or decision, each candidate is added to R or discarded. □

In general, there is a subtype relation which should be satisfied by G_{LF} but is not necessarily obtained by the above heuristics. A human translator must find such a subtype

Table 8 Subtype relation for restrictions.

```
A[abort_accept_spdu,1] <==> A[abort_spdu,1]
A[abort_spdu,1] <==> A[accept_spdu,1]
A[accept_spdu,1] <==> A[major_sync_point_spdu,1]
A[s_connect_accept_confirm,1] <==> A[s_connect_accept_response,1]
A[s_connect_accept_response,1] <==> A[s_connect_request,1]
A[s_connect_request,1] <==> A[s_sync_major_indication,1]
A[s_sync_major_indication,1] <==> A[s_sync_major_request,1]
```

relation. If G and R are simplified by the method stated in the next section, finding such a subtype relation becomes easier.

5. Simplification of the Grammar

In the last phase, $G = (N, T, P)$ and R are simplified based on structural equivalence [10] of cfg's. For simplicity, in the following definitions on structural equivalence, each element $A \overset{*}{\Rightarrow} B$ in R is regarded as a production rule $A \to B$ to be in P.

First, we define the notion of structural equivalence.

Definition 1: Let $\mathcal{T}_G$ denote the set of all parse trees generated by cfg G. For any $t = (V, E) \in \mathcal{T}_G$, define two mappings $h_V : V \to V$ and $h_E : E \to E$ as follows:

$$h_V(v) = \begin{cases} h_V(v') & \text{if } v \text{ has exactly one child node } v' \text{ and} \\ & \quad v' \text{ is an internal node,} \\ v & \text{otherwise,} \end{cases}$$

$$h_E((v_1, v_2)) = \begin{cases} (h_V(v_1), h_V(v_2)) & \text{if } h_V(v_1) \neq h_V(v_2), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In addition, let

$$h(t) = (\{h_V(v) \mid v \in V\}, \{h_E(e) \mid e \in E\}),$$

and

$$h(\mathcal{T}_G) = \{h(t) \mid t \in \mathcal{T}_G\}.$$

Table 9 Subtype relation obtained by Heuristics 1.

```
SPDU ===> A[incoming,1]
SPDU ===> nt1
SPDU ===> A[sending,2]
SPDU ===> A[valid,1]
SSprm ===> nt2
A[assignment,1] ===> [transport_connection,1]
A[incoming,1] ===> [valid,1]
nt1 ===> SPDU
nt1 ===> A[incoming,1]
nt1 ===> A[sending,2]
nt1 ===> A[valid,1]
nt2 ===> SSprm
A[results_in,1] ===> A[incoming,1]
A[results_in,1] ===> A[valid,1]
A[transport_connection,1] ===> A[assignment,1]
A[valid,1] ===> A[incoming,1]

nt1 = {A[abort_accept_spdu,1], A[abort_spdu,1],
       A[accept_spdu,1], A[major_sync_point_spdu,1]}
nt2 = {A[s_connect_accept_confirm,1], A[s_connect_accept_response,1],
       A[s_connect_request,1], A[s_sync_major_indication,1],
       A[s_sync_major_request,1]}
```

Two cfg's G_1 and G_2 are said to be structurally equivalent, if $h(\mathcal{T}_{G_1}) = h(\mathcal{T}_{G_2})$ when the labels of the internal nodes are ignored. $\square$

Intuitively, structural equivalence of G_1 and G_2 means that the set of all parse trees generated by G_1 is equal to the one generated by G_2 when the labels of the internal nodes and applications of unit productions ($A \to A'$) are ignored.

Ref. [5] presents an algorithm which decides whether two arbitrary parenthesis grammars are weakly equivalent. For a given cfg G, the equivalence classes of the nonterminals based on structural equivalence (defined below) are obtained by using this algorithm. G is simplified by replacing each nonterminal with its equivalence class.

Definition 2: Let $\mathcal{T}_G[A]$ denote the set of parse trees with internal nodes labelled A. Let $\mathcal{T}_G[A \leftarrow \Omega] = \{t' \mid t'$ is obtained from some $t \in \mathcal{T}_G[A]$ by replacing one subtree of t whose root is A with a single node labelled $A\}$. Define h in the same way as defined in Definition 1.

Two nonterminals A and B of G are said to be structurally equivalent, if

$$h(\mathcal{T}_G[A \leftarrow \Omega]) = h(\mathcal{T}_G[B \leftarrow \Omega])$$

when the labels of the internal nodes of $h(\mathcal{T}_G[A \leftarrow \Omega])$ and $h(\mathcal{T}_G[B \leftarrow \Omega])$ are ignored. $\square$

The algorithm proposed in Ref. [5] takes $O(2^{2^r})$ time for a cfg with r nonterminals. However, only prefix function symbols are defined in a grammar to be constructed by our method. This improves the time complexity of the algorithm. For example, we assume that G satisfies the following condition:

For any function symbol f, there is at most one nonterminal C such that

$$C \underset{G}{\Rightarrow} f(A_{(f,1)}, A_{(f,2)}, \ldots, A_{(f,n)}).$$

Then, by Definition 2, the following property holds:

Nonterminals A and B are structurally equivalent if and only if for any function symbol f and its argument position i,

$$A_{(f,i)} \overset{*}{\underset{G}{\Rightarrow}} A \quad \text{iff} \quad A_{(f,i)} \overset{*}{\underset{G}{\Rightarrow}} B. \tag{6}$$

By using this property, G can be simplified in polynomial time of the description size of G.

Augmentation of G and R follows this simplification if necessary. As stated in the end of the previous section, and as also illustrated in the following example, the augmentation becomes easier for a human translator.

Example 8: Suppose that every subtype relation in Table 9 is added to R by a human translator. The result of simplification of R is shown in Fig. 2, where an arc from A' to A means that A is a subtype of A'. Each `nt*` is an equivalence class of nonterminals based on structural equivalence, and its members are shown in Table 10.

The graph shown in Fig. 2 is symmetric when the arc from `nt8` to `nt12` is ignored. Consider the meaning of `nt12`. It is a subtype of `nt8`, which contains `SPDU`, and it contains `A[sending,2]`, i.e., the data type of the objects of gerund "sending." It follows that `nt12` is a data type denoting SPDUs to be transmitted. On the other hand, `nt3` is a subtype of both `nt8` and `nt10`. Since `nt10` contains `A[results_in,2]`, i.e., the data type of the objects of phrasal verb "results in," `nt8` is also a data type denoting SPDUs to be transmitted. Therefore, it seems appropriate for the following subtype relation

```
nt3 <==> nt12
```

to be added to R.

As a result, the subtype relation shown in Fig. 3 is obtained, where the name of each nonterminal is replaced with a mnemonic one (see Table 11). □

When a human translator considers G and R to be appropriate, each $A \stackrel{*}{\Rightarrow} B$ in R is transformed into a production $A \rightarrow B$ of G. Then, G_{LF} is constructed as the component-wise union of G and G_0, where G_0 is the specification of primitive data types introduced in Sect. 2.

Example 9: From the six sentences in Table 2, the grammar shown in Table 12 is constructed as an appropriate one. □

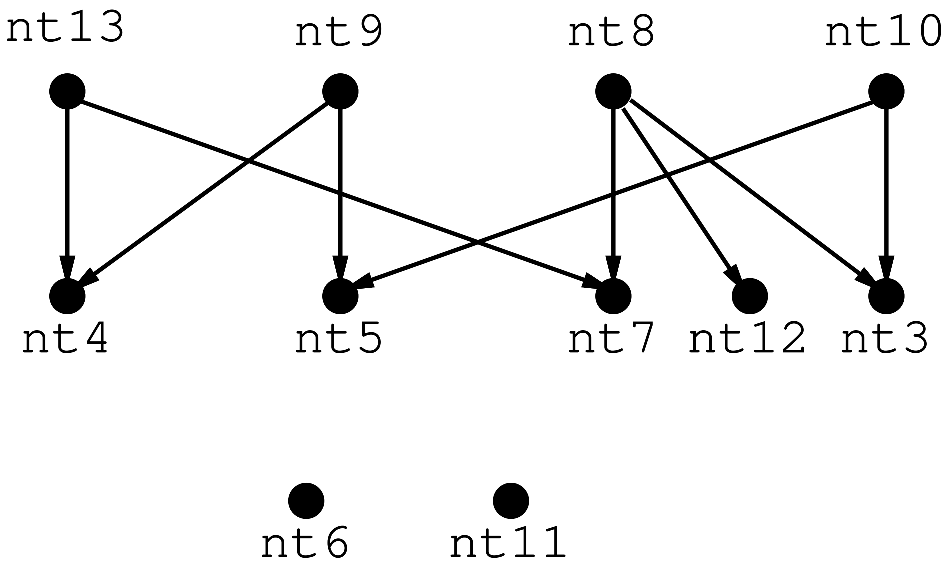

Fig. 2 Result of simplification of the subtype relation.

Table 10 Equivalence classes in Fig. 2.

```
nt3  = {A[_x3], A[_x10]}
nt4  = {A[_x2], A[_x4], A[_x11]}
nt5  = {A[_x5], A[_x12]}
nt6  = {A[_x1], A[assignment,1], A[transport_connection,1]}
nt7  = {A[_x6], A[_x9], A[_x13], A[incoming,1], A[valid,1]}
nt8  = {SPDU, A[abort_accept_spdu,1], A[abort_spdu,1],
        A[accept_spdu,1], A[major_sync_point_spdu,1]}
nt9  = {SSprm, A[s_connect_accept_confirm,1],
        A[s_connect_accept_response,1], A[s_connect_request,1],
        A[s_sync_major_indication,1], A[s_sync_major_request,1]}
nt10 = {A[assignment], A[results_in,2], A[sending]}
nt11 = {A[_x8], A[sending,1]}
nt12 = {A[_x7], A[sending,2]}
nt13 = {A[results_in,1]}
```

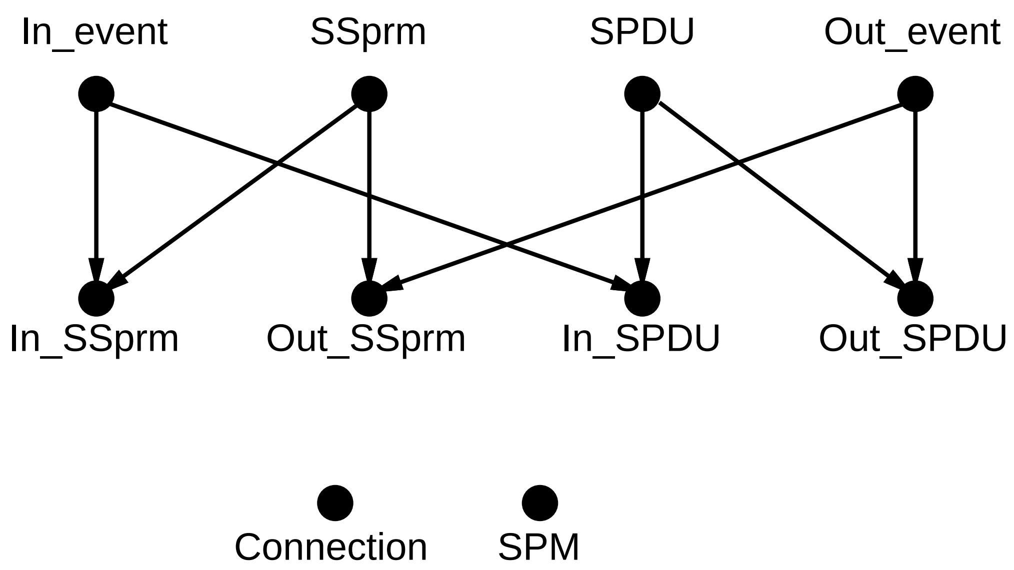

Fig. 3 Appropriate subtype relation.

Table 11 Data types in Fig. 3.

- In_event: Input events.
- Out_event: Output events.
- SPDU: Session protocol data units.
- SSprm: Session service primitives.
- In_SPDU: Received session protocol data units.
- In_SSprm: Received session service primitives.
- Out_SPDU: Transmitted session protocol data units.
- Out_SSprm: Transmitted session service primitives.
- Connection: Connections.
- SPM: Session protocol machines.

Table 12   Appropriate grammar.

```
Bool ---> abort_accept_spdu(SPDU)
Bool ---> abort_spdu(SPDU)
Bool ---> accept_spdu(SPDU)
Bool ---> incoming(In_SPDU)
Bool ---> major_sync_point_spdu(SPDU)
Bool ---> results_in(In_event,Out_event)
Bool ---> s_connect_accept_confirm(SSprm)
Bool ---> s_connect_accept_response(SSprm)
Bool ---> s_connect_request(SSprm)
Bool ---> s_sync_major_indication(SSprm)
Bool ---> s_sync_major_request(SSprm)
Bool ---> transport_connection(Connection)
Bool ---> valid(In_SPDU)
Out_event ---> assignment(Connection)
Out_event ---> sending(SPM,Out_SPDU)
In_event ---> In_SPDU
In_event ---> In_SSprm
Out_event ---> Out_SPDU
Out_event ---> Out_SSprm
SPDU ---> In_SPDU
SPDU ---> Out_SPDU
SSprm ---> In_SSprm
SSprm ---> Out_SSprm
```

6. Evaluation of the Assignment Method

Ref. [9] proposes a method of analyzing anaphoric bindings in a natural language specification. In the method, type information is used for selecting candidates for the antecedent of an anaphoric phrase. For the method to work well, words and phrases must be appropriately classified. In this section, we show an example in which ambiguity of anaphoric bindings is solved by G_{LF} constructed by our method.

Example 10: Consider the following sentences in Ref. [3]:

> A valid incoming ABORT SPDU results in sending an ABORT ACCEPT SPDU. This SPDU is sent on the transport normal flow.

From these two sentences, expressions

$$\mathsf{result\_in}(\bar{x}_1, \mathsf{sending}(\bar{x}_2)),\ \mathsf{be\_sent\_on}(\bar{x}_?, \bar{x}_3)$$

are obtained, where $\bar{x}_1$ corresponds to "ABORT SPDU," $\bar{x}_2$ does to "ABORT ACCEPT SPDU," $\bar{x}_?$ does to "this SPDU," and $\bar{x}_3$ does to "the transport normal flow."

Suppose that $\bar{x}_1$, $\bar{x}_2$ and $\bar{x}_?$ are of type SPDU. By only using this type information, it can not be determined whether "this SPDU" refers to "a valid incoming ABORT SPDU" or "an ABORT ACCEPT SPDU."

Now suppose that G_{LF} contains the production rules shown in Table 12 and $\mathsf{Bool} \to \mathsf{be\_sent\_on}(\mathsf{Out\_event}, \mathsf{Flow})$. Also suppose that $L[\mathsf{In\_event}] \cap L[\mathsf{Out\_event}] = \emptyset$. Then,

- the data type of $\bar{x}_1$ is $\mathsf{In\_SPDU}$,
- the data type of $\bar{x}_2$ is $\mathsf{Out\_SPDU}$, and
- the data type of $\bar{x}_?$ is $\mathsf{Out\_SPDU}$.

Assume that "this SPDU" refers to "ABORT SPDU." Then the variables corresponding to these two phrases are identical, i.e., $\bar{x}_? = \bar{x}_1$, and the type of $\bar{x}_1$ is the greatest type among subtypes of both $\mathsf{In\_SPDU}$ and $\mathsf{Out\_SPDU}$. On the other hand, $L[\mathsf{In\_SPDU}] \cap L[\mathsf{Out\_SPDU}] = \emptyset$ since $L[\mathsf{In\_event}] \cap L[\mathsf{Out\_event}] = \emptyset$, and hence such a data type does not exist. Thus, it can be determined that "this SPDU" can not refer "ABORT SPDU."

□

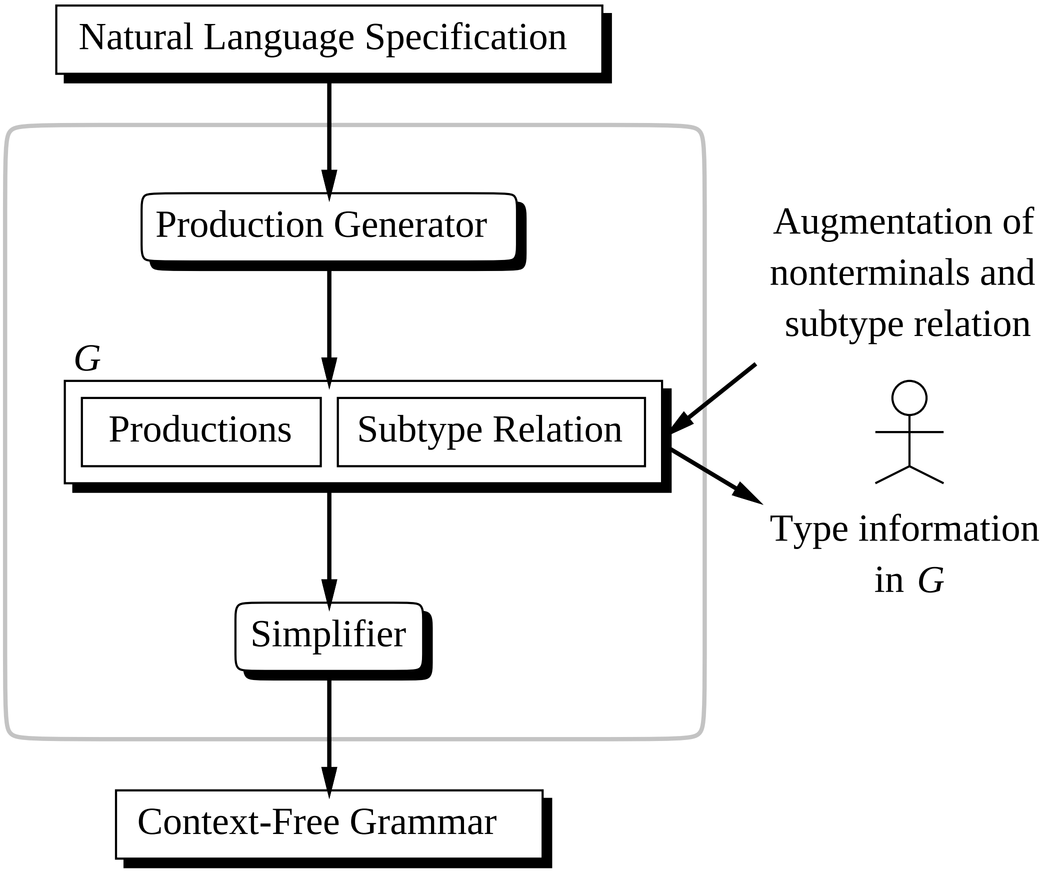

Fig. 4 Assignment system.

7. Assignment System

Based on the method proposed in this paper, we implemented a prototype system (see Fig. 4) on SPARCstation IPX. Production Generator consists of about 200 DCG [7] rules and about 660 Prolog rules. Simplifier is written in C and the size is about 230 lines. We also implemented a program in Tcl/Tk which shows a given subtype relation graphically.

By using the prototype system, data types were assigned to words in a part of the OSI session protocol specification [3]. The part consists of 39 sentences including words "result" or "wait," or describing operations on the registers of a session protocol machine. The result of applying the method to the 39 sentences is shown in Table 13, where a couple of subtype relations $A \stackrel{*}{\Rightarrow} B$ and $B \stackrel{*}{\Rightarrow} A$ added simultaneously are counted as one subtype relation.

Table 13 Result of the assignment.

	$\|N\|$	$\|R\|$
Generated at Phase 1	169	180
Added based on		
Boolean	1	15
Grouping	11	111
Meanings of words	0	14
Obtained by removing $\overset{*}{\Leftrightarrow}$	127	248
Added based on		
Heuristics 1[†]	0	72
Obtained by removing $\overset{*}{\Leftrightarrow}$	99	121
Added based on		
Heuristics 2[†]	0	47
Obtained by removing $\overset{*}{\Leftrightarrow}$	44	37
Obtained by simplification	27	23
Added based on		
Meanings of words	0	3
Obtained by removing $\overset{*}{\Leftrightarrow}$	24	20

[†]The numbers of candidates by Heuristics 1 and 2 were 84 and 49, respectively.

Since the prototype system automatically substitutes `Bool` for the type of the return value of the function denoting the restriction of each noun, subtype relations corresponding to such replacements are not counted in Table 13.

8. Conclusions

This paper proposed a method of assigning data types to words in a natural language specification. The result of applying this method to a part of the OSI session protocol specification was also presented.

In recent years, various object-oriented analysis methods have been proposed [1]. In many of them, for example in OMT [8], candidates of objects and methods are extracted by considering the part of speech of each word in a natural language specification. Then, the candidates are refined and classified by considering the meanings of the words in the natural language specification. As stated in Sect. 1, classifying objects and methods is time-consuming when an input natural language specification is large. Moreover, an appropriate classification often seems to be obtained after a human analyzer understands the natural language specification by trial and error. Therefore, it is desirable that the classification process be semi-automated. Our semi-automated method will contribute toward improving those object-oriented analysis methods.

Acknowledgments

The authors wish to thank Professor Kenichi Taniguchi of Osaka University and Professor Minoru Ito of Nara Institute of Science and Technology for their invaluable suggestions and comments.

References

[1] Honiden, S. and Yamashiro, A., “Object-Oriented Analysis and Design,” Journal of IPS Japan, vol. 35, no. 5, pp. 392–401, May 1994 (in Japanese).

[2] Ishihara, Y., Seki, H., Kasami, T., Shimabukuro, J. and Okawa, K., “A Translation Method from Natural Language Specifications of Communication Protocols into Algebraic Specifications Using Contextual Dependencies,” IEICE Trans. Inf. & Syst., vol. E76-D, no. 12, pp. 1479–1489, Dec. 1993.

[3] ISO, “Basic Connection Oriented Session Protocol Specification,” ISO 8327, 1987.

[4] Kasami, T., Taniguchi, K., Sugiyama, Y. and Seki, H., “Principles of Algebraic Language ASL/∗,” Trans. IECE Japan, vol. J69-D, no. 7, pp. 1066–1074, July 1986 (in Japanese), also in Systems and Computers in Japan, vol. 18, no. 7, pp. 11–20, July 1987.

[5] McNaughton, R., “Parenthesis Grammar,” Journal of ACM, vol. 14, no. 3, pp. 490–500, July 1967.

[6] Ohsaki, A., Ishihara, Y., Seki, H. and Kasami, T., “Generation of Signatures of Functions in Translation from Natural Language Specifications into Algebraic Specifications,” IPSJ SIG Notes, SE-96-10, Jan. 1994 (in Japanese).

[7] Pereira, F. C. N. and Warren, D. H. D., “Definite Clause Grammars for Language Analysis,” Artificial Intelligence, vol. 13, pp. 231–278, 1980.

[8] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W., “Object-Oriented Modeling and Design,” Prentice-Hall, 1990.

[9] Seki, H., Kasami, T., Nabika, E. and Matsumura, T., “A Method for Translating Natural Language Program Specifications into Algebraic Specifications,” Trans. IEICE Japan, vol. J74-D-I, no. 4, pp. 283–295, Apr. 1991 (in Japanese).

[10] Taniguchi, K. and Kasami, T., “Reduction of Context-Free Grammars,” Trans. IECE Japan, vol. 52-C, no. 12, pp. 827–834, Dec. 1969 (in Japanese).