ソフトウェア工学 99－1
（１９９４．７．７）

LOTOS 風言語の代数的意味定義とその応用

石原 靖哲　関 浩之　嵩 忠雄
奈良先端科学技術大学院大学　情報科学研究科

本稿では, BE program と呼ばれる代数的仕様の部分クラスを与える. BE program の構文は LOTOS の構文と似ており, BE program の意味は, 有限個のレジスタと有限個の非有界 I/O バッファをもつインタプリタ (BE interpreter と呼ぶ) の振舞いとして定義される. BE interpreter の振舞いは状態遷移モデルに基づく代数的公理で記述されており, C 言語等で容易に実現することができる. また, 本稿では, 通信プロトコルの自然語仕様から得られた代数的仕様を, BE program で実現する手法についても述べる. これにより, 自然語仕様から実行可能なプログラムへの一貫した変換が可能となる.

An Algebraic Definition of a LOTOS-Like Language and Its Application

Yasunori ISHIHARA　Hiroyuki SEKI　Tadao KASAMI
Graduate School of Information Science
Nara Institute of Science and Technology

This paper proposes a subclass of algebraic specifications called BE programs. The syntax of BE programs resembles the syntax of LOTOS, and the semantics of BE programs is defined as a behavior of an interpreter, called a BE interpreter, which has a finite number of registers and unbounded I/O buffers. BE interpreters can be easily implemented by C. This paper also proposes a method of implementing a subclass of algebraic specifications, which are derived from natural language specifications of communication protocols, by BE programs. Thus, one can translate a natural language specification of communication protocols into an executable program.

1 Introduction

To raise effectiveness of software development processes and reliability of programs, various formal specification methods have been proposed and studied. Algebraic specification methods [1], [2] are useful and powerful because of the following reasons:

1. Abstract data types can be defined simply in algebraic specifications;
2. Formal semantics of a specification is simply provided by axioms (equations); and
3. One can write a specification which has arbitrary structure and arbitrary degree of abstraction.

Using algebraic specification methods, we define a class of interpreters (machines), called BE interpreters, each of which has a finite number of registers and unbounded I/O buffers. A BE interpreter performs three kind of atomic actions: input from a buffer, output to a buffer, and calculation using its registers. An input program for a BE interpreter, called a BE program, specifies the order of actions by means of such operators as action-prefix, choice, conditional, and so on. The syntax of BE programs is also defined within the framework of algebraic specification. And the semantics of BE programs, i.e., the behavior of BE interpreters, is defined by axioms based on a state transition model. Therefore, each BE interpreter can be easily implemented.

In Refs. [3] and [6], we proposed a translation method from natural language specifications of communication protocols into algebraic specifications. A specification such as a protocol specification defines valid sequences of actions performed by a protocol machine. In the method, the valid sequences of actions are represented by an axiom whose left-hand side is a formula in first-order predicate logic. In this paper, we present a method of implementing such axioms by BE programs.

2 Algebraic Specification Language ASL

In this paper, we adopt ASL [5] as an algebraic specification language. A specification in ASL is a pair $SPEC = (G, AX)$ of a context-free grammar G and a set AX of axioms. G specifies the set of expressions and their syntax, and AX specifies their semantics. Let $G = (N, T, P)$ where N, T and P are sets of nonterminals, terminals and productions, respectively. For a nonterminal $D \in N$, let $L_G[D]$ denote the set of terminal strings derived from D in G, and let $L_G = \bigcup_{D \in N} L_G[D]$. An element in L_G is called an expression (in the specification *SPEC*). N corresponds to the set of sorts (data types); A nonterminal D is sometimes called "data type D" and an expression in $L_G[D]$ may be called "an expression of type D."

An axiom is a pair $l == r$ 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 $D_{\bar{x}}$ is associated (declared by "$\bar{x} : D_{\bar{x}}$" in the specification), and an arbitrary expression in $L_G[D_{\bar{x}}]$ can be substituted into $\bar{x}$. The least congruence relation that satisfies all the axioms in AX is denoted by $\equiv_{SPEC}$. See Ref. [5] for details.

Table 1: Specification of sequences.

- Production schemata:

$$\begin{aligned} \mathsf{Seq\_}D &\rightarrow \lambda, \\ \mathsf{Seq\_}D &\rightarrow \mathsf{Seq\_}D \cdot D, \\ D &\rightarrow \mathsf{head}(\mathsf{Seq\_}D), \\ \mathsf{Seq\_}D &\rightarrow \mathsf{tail}(\mathsf{Seq\_}D), \\ \mathsf{Bool} &\rightarrow \mathsf{member}(D, \mathsf{Seq\_}D). \end{aligned}$$

- Axioms:

$$\begin{aligned} &\bar{x}_{\mathsf{seq}} : \mathsf{Seq\_}D,\ \bar{x}, \bar{x}' : D \\ \mathsf{head}(\bar{x}_{\mathsf{seq}} \cdot \bar{x}) &== \mathsf{if}\ \bar{x}_{\mathsf{seq}} = \lambda\ \mathsf{then}\ \bar{x} \\ &\qquad\qquad \mathsf{else}\ \mathsf{head}(\bar{x}_{\mathsf{seq}}), \\ \mathsf{tail}(\lambda) &== \lambda, \\ \mathsf{tail}(\bar{x}_{\mathsf{seq}} \cdot \bar{x}) &== \mathsf{if}\ \bar{x}_{\mathsf{seq}} = \lambda\ \mathsf{then}\ \lambda \\ &\qquad\qquad \mathsf{else}\ \mathsf{tail}(\bar{x}_{\mathsf{seq}}) \cdot \bar{x}, \\ \mathsf{member}(\bar{x}, \lambda) &== \mathsf{false}, \\ \mathsf{member}(\bar{x}, \bar{x}_{\mathsf{seq}} \cdot \bar{x}') &== \mathsf{if}\ \bar{x} = \bar{x}'\ \mathsf{then\ true} \\ &\qquad\qquad \mathsf{else}\ \mathsf{member}(\bar{x}, \bar{x}_{\mathsf{seq}}). \end{aligned}$$

In this paper, we presuppose a fixed specification $SPEC_0 = (G_0, AX_0)$, $G_0 = (N_0, T_0, P_0)$ of primitive data types (e.g., integer, Boolean, set, and so on), which defines the data types of the contents of the registers and I/O buffers of a BE interpreter. We assume that $SPEC_0$ supports the following data types:

1. Boolean; Let Bool be a nonterminal which generates Boolean expressions.
2. Sequence; Let Seq_ be a constructor on data types to support sequences of a given data type, i.e., for any data type D, Seq_D generates sequences of expressions of type D. Formally, $SPEC_0$ has the production schemata and axioms shown in Table 1, where $\lambda, \cdot, \mathsf{head}, \mathsf{tail}, \mathsf{member} \in T_0$. Constant function λ denotes the empty sequence and function "$\cdot$" denotes the concatenation operation. For a given sequence, head returns the first element and tail returns the sequence obtained by eliminating the first element. Predicate member is true if and only if the first parameter is an element of the second parameter.

3 BE Programs

As stated in Sect. 1, a BE interpreter has registers and I/O buffers, and performs three kind of atomic actions. The syntax and semantics of a BE program are defined to meet the following requirements:

(a) One can describe all the registers and I/O buffers of a BE interpreter;
(b) One can use primitive data types as ones of contents of the registers and I/O buffers;
(c) One can describe all the actions performed by a BE interpreter; and
(d) One can specify the order of performance of actions.

In this section, we restate these requirements formally, i.e., we describe the conditions which a BE program $SPEC_{\mathrm{PRG}} = (G_{\mathrm{PRG}}, AX_{\mathrm{PRG}})$, $G_{\mathrm{PRG}} = (N_{\mathrm{PRG}}, T_{\mathrm{PRG}}, P_{\mathrm{PRG}})$ has to meet.

First, for the requirement (a), we introduce two data types Reg and Buf as follows:

1. Reg $\in N_{\text{PRG}}$ generates names of registers of a BE interpreter. Each element of $L_{G_{\text{PRG}}}[\text{Reg}]$ is a terminal symbol. We mean $L_{G_{\text{PRG}}}[\text{Reg}]$ by *REG*.
2. Buf $\in N_{\text{PRG}}$ generates names of I/O buffers. Each element of $L_{G_{\text{PRG}}}[\text{Buf}]$ is also a terminal symbol. We mean $L_{G_{\text{PRG}}}[\text{Buf}]$ by *BUF*.

We assume that $REG \cap BUF = \emptyset$.

Secondly, for the requirement (b), $SPEC_{\text{PRG}}$ must satisfy the following condition:

3. $SPEC_{\text{PRG}} \supset SPEC_0$ (pair-wise containment).
4. For each $reg \in REG$, there is a unique nonterminal symbol $D_{reg} \in N_0$ such that $D_{reg} \to reg \in P_{\text{PRG}}$. D_{reg} is denoted by $type[reg]$.
5. For each $buf \in BUF$, there is a unique nonterminal symbol $D_{buf} \in N_0$ such that $\text{Seq\_}D_{buf} \to buf \in P_{\text{PRG}}$. D_{buf} is denoted by $type[buf]$.

Thirdly, for the requirement (c), we introduce the following data type:

6. Action $\in N_{\text{PRG}}$ generates actions. For each $buf \in BUF$ and $reg \in REG$, the following productions are in P_{PRG}:
$$\begin{aligned}\text{Action} &\to \text{in}(buf, reg),\\ \text{Action} &\to \text{out}(buf, reg),\\ \text{Action} &\to \text{set}(reg \leftarrow D_{reg}),\end{aligned}$$
where in, out, set, $\leftarrow \in T_{\text{PRG}}$, $type[buf] = type[reg]$, and $D_{reg} = type[reg]$. in(buf, reg) denotes that a BE interpreter receives a data from buffer *buf* and the data is stored in register *reg*. out(buf, reg) denotes an action that a BE interpreter transmits a data stored in register *reg* to buffer *buf*. set$(reg \leftarrow t)$ denotes an assignment of the value of a term t to register *reg* (the value of a term is formally defined in Sect. 4).

Lastly, for the requirement (d), we introduce behavior expressions, which specifies the order of performance of actions. Some behavior expressions are associated with behavior identifiers so that a behavior expression can refer (call) another behavior expression, i.e., a behavior identifier corresponds to a procedure name. The syntax of behavior expressions is defined as follows:

7. B_id $\in N_{\text{PRG}}$ generates behavior identifiers. There are productions of the following form:
$$\text{B\_id} \to \pi,$$
where $\pi \in T_{\text{PRG}}$ is a behavior identifier.
8. B_exp $\in N_{\text{PRG}}$ generates behavior expressions. The following productions are in P_{PRG}:
$$\begin{aligned}&\text{B\_exp} \to \text{stop},\\ &\text{B\_exp} \to \text{B\_id},\\ &\text{B\_exp} \to \text{Action}; \text{B\_exp},\\ &\text{B\_exp} \to (\text{B\_exp} \,\Box\, \text{B\_exp}),\\ &\text{B\_exp} \to (\text{B\_exp}|\text{Seq\_Action}|\text{B\_exp}),\\ &\text{B\_exp} \to [\text{Bool}] \text{->} \text{B\_exp},\\ &\text{B\_exp} \to (\text{B\_exp} \gg \text{Seq\_Action} \gg \text{B\_exp}),\\ &\text{B\_exp} \to (\text{B\_exp} \,[>\, \text{Seq\_Action} \,[>\, \text{B\_exp}),\end{aligned}$$
where stop, ;, $\Box$, |, [,], ->, (,), $\gg$, [> $\in T_{\text{PRG}}$.

Table 2: Meanings of operators.

- stop means that no actions are performed, i.e., a BE interpreter which executes it comes to a deadlock.
- Execution of a behavior identifier π is equivalent to execution of the behavior expression which is associated with π.
- Action-prefix: $a; B$ specifies that a BE interpreter performs action a, then executes behavior expression B.
- Choice: $(B_1 \Box B_2)$ specifies that a BE interpreter executes either B_1 or B_2 nondeterministically.
- Parallel composition: $(B_1|\lambda \cdot a_1 \cdots a_n|B_2)$ specifies that a BE interpreter executes behavior expressions B_1 and B_2 in a "time sharing" manner. Here, each action a_i $(1 \le i \le n)$ must be simultaneously performed in executions of B_1 and B_2.
- Conditional: $[p]$ -> B specifies that a BE interpreter executes behavior expression B if predicate p holds, and comes to a deadlock otherwise.
- Enabling: $(B_1 \gg \lambda \cdot a_1 \cdots a_n \gg B_2)$ specifies that a BE interpreter executes behavior expression B_1 first. When some action a_i $(1 \le i \le n)$ is performed during execution of B_1, the interpreter begins to execute behavior expression B_2.
- Disabling: $(B_1 \,[>\, \lambda \cdot a_1 \cdots a_n \,[>\, B_2)$ specifies that a BE interpreter executes behavior expression B_1, and the interpreter can nondeterministically begin to execute behavior expression B_2 until some action a_i $(1 \le i \le n)$ is performed during execution of B_1.

In Table 2, we present the intuitive meanings of the operators used in behavior expressions. In Sect. 4, we define the formal semantics as the behavior of a BE interpreter.

Now we introduce a predicate := which associates a behavior expression with a behavior identifier.

9. There is a production
$$\text{Bool} \to \text{B\_id} := \text{B\_exp}$$
in P_{PRG}, and there are one or more axioms
$$\pi := B == \text{true}$$
in AX_{PRG}, where $:= \in T_{\text{PRG}}$, $\pi \in L_{G_{\text{PRG}}}[\text{B\_id}]$, and $B \in L_{G_{\text{PRG}}}[\text{B\_exp}]$. $\pi := B \equiv_{SPEC_{\text{PRG}}} \text{true}$ means that π is defined as B in $SPEC_{\text{PRG}}$. We call an expression in the form of $\pi := B$ a behavior definition (of π).

Among the behavior expressions defined by operator :=, exactly one behavior expression must be specified as the main (top level) behavior expression, i.e., the one which should be executed first by a BE interpreter.

10. There is a production
$$\text{Bool} \to \text{main(B\_id)}$$
in P_{PRG}, and there is exactly one axiom
$$\text{main}(\pi) == \text{true}$$
in AX_{PRG}, where main $\in T_{\text{PRG}}$ and $\pi \in L_{G_{\text{PRG}}}[\text{B\_id}]$. $\text{main}(\pi) \equiv_{SPEC_{\text{PRG}}} \text{true}$ means that π is the main behavior expression.

To execute the main behavior expression, the initial values of the registers and I/O buffers must be specified:

11. For each $y \in REG \cup BUF$, the following production is in P_{PRG}:
$$D_y \to \text{initial}(y),$$

where $\mathsf{initial} \in T_{\mathrm{PRG}}$, $D_y = type[y]$ if $y \in REG$, and $D_y = \mathsf{Seq\_}type[y]$ if $y \in BUF$. Moreover, for each $y \in REG \cup BUF$, there is exactly one axiom

$$\mathsf{initial}(y) == c_y$$

in AX_{PRG}, where $c_y \in L_{G_{\mathrm{PRG}}}[type[y]]$ if $y \in REG$ and $c_y \in L_{G_{\mathrm{PRG}}}[\mathsf{Seq\_}type[y]]$ if $y \in BUF$. $\mathsf{initial}(y) \equiv_{SPEC_{\mathrm{PRG}}} c_y$ means that the initial value of y is c_y.

4 BE Interpreters

In this section, we define the semantics of BE programs in terms of the behavior of a BE interpreter. Let $SPEC_{\mathrm{INT}} = (G_{\mathrm{INT}}, AX_{\mathrm{INT}})$, $G_{\mathrm{INT}} = (N_{\mathrm{INT}}, T_{\mathrm{INT}}, P_{\mathrm{INT}})$ be a BE interpreter specification, and $SPEC_{\mathrm{PRG}} = (G_{\mathrm{PRG}}, AX_{\mathrm{PRG}})$, $G_{\mathrm{PRG}} = (N_{\mathrm{PRG}}, T_{\mathrm{PRG}}, P_{\mathrm{PRG}})$ a BE program.

First, we assume that $G_{\mathrm{INT}} \supseteq G_{\mathrm{PRG}}$ (pair-wise containment). By this assumption, it can be considered that $SPEC_{\mathrm{INT}} \cup SPEC_{\mathrm{PRG}}$ (pair-wise union) specifies the behavior of a BE interpreter when $SPEC_{\mathrm{PRG}}$ is given as its input program. We mean $L_{G_{\mathrm{INT}}}[\mathsf{Reg}]$ and $L_{G_{\mathrm{INT}}}[\mathsf{Buf}]$ by REG and BUF, respectively.

Next, we define the semantics of the operators used in behavior expressions. To do this, we introduce a quadruple relation $EXEC$. Let $B, B' \in L_{G_{\mathrm{INT}}}[\mathsf{B\_exp}]$, $p \in L_{G_{\mathrm{INT}}}[\mathsf{Bool}]$, and $a \in L_{G_{\mathrm{INT}}}[\mathsf{Action}]$. $\langle B, p, a, B' \rangle \in EXEC$ means that "Suppose that a BE interpreter is about to execute behavior expression B, and that the values of the registers and I/O buffers of the BE interpreter satisfy predicate p. Then, the BE interpreter is allowed to perform action a, and then executes behavior expression B' after a." In $SPEC_{\mathrm{INT}}$, relation $EXEC$ is represented by a predicate exec. We introduce a production

$$\mathsf{Bool} \rightarrow \mathsf{exec}(\mathsf{B\_exp}, \mathsf{Bool}, \mathsf{Action}, \mathsf{B\_exp})$$

into P_{INT}, and the axioms shown in Table 3 into AX_{INT}.

Now, we introduce $\mathsf{State} \in N_{\mathrm{INT}}$, which is a data type representing states of the BE interpreter. The productions whose left-hand side is State are as follows:

$$\mathsf{State} \rightarrow \mathsf{s}_{\mathsf{init}},$$
$$\mathsf{State} \rightarrow \delta(\mathsf{State}, \mathsf{Action}),$$

where $\mathsf{s}_{\mathsf{init}}, \delta \in T_{\mathrm{INT}}$. $\mathsf{s}_{\mathsf{init}}$ denotes the initial state of the BE interpreter, and $\delta(s, a)$ denotes the state immediately after action a is performed at state s.

By using the notion of the states of a BE interpreter, we define the semantics of each action a as relation between the values of the registers and I/O buffers before a is performed and the values after a is performed. To express this relation in $SPEC_{\mathrm{INT}}$, we introduce $\mathsf{val} \in T_{\mathrm{INT}}$ such that for each $D \in N_0$, a production

$$D \rightarrow \mathsf{val}(D, \mathsf{State})$$

is in P_{INT}. For any expression t which includes some of members of $REG \cup BUF$, $\mathsf{val}(t, s)$ denotes the value of t at state s. The semantics of the actions is defined by the axioms shown in Table 4.

Lastly, a BE interpreter has to remember what behavior expression it is executing. This is easily achieved by

Table 3: Axioms on exec.

$\bar{B}, \bar{B}', \bar{B}_1, \bar{B}'_1, \bar{B}_2, \bar{B}'_2 : \mathsf{B\_exp}$, $\bar{a} : \mathsf{Action}$,
$\bar{p}, \bar{p}', \bar{p}_1, \bar{p}_2 : \mathsf{Bool}$, $\bar{\pi} : \mathsf{B\_id}$, $\bar{A} : \mathsf{Seq\_Action}$

- Action-prefix:

$$\mathsf{exec}(\bar{a}; \bar{B}, \mathsf{true}, \bar{a}, \bar{B}) == \mathsf{true}.$$

- Choice:

$$\begin{aligned}&\mathsf{exec}(\bar{B}_1, \bar{p}_1, \bar{a}, \bar{B}'_1) \supset \\ &\quad \mathsf{exec}((\bar{B}_1 \,\square\, \bar{B}_2), \bar{p}_1, \bar{a}, \bar{B}'_1) == \mathsf{true}, \\ &\mathsf{exec}(\bar{B}_2, \bar{p}_2, \bar{a}, \bar{B}'_2) \supset \\ &\quad \mathsf{exec}((\bar{B}_1 \,\square\, \bar{B}_2), \bar{p}_2, \bar{a}, \bar{B}'_2) == \mathsf{true}.\end{aligned}$$

- Instantiation:

$$\begin{aligned}&((\bar{\pi} := \bar{B}) \wedge \mathsf{exec}(\bar{B}, \bar{p}, \bar{a}, \bar{B}')) \supset \\ &\qquad \mathsf{exec}(\bar{\pi}, \bar{p}, \bar{a}, \bar{B}') == \mathsf{true}.\end{aligned}$$

- Parallel composition:

$$\begin{aligned}&(\mathsf{exec}(\bar{B}_1, \bar{p}_1, \bar{a}, \bar{B}'_1) \wedge \neg\mathsf{member}(\bar{a}, \bar{A})) \supset \\ &\quad \mathsf{exec}((\bar{B}_1|\bar{A}|\bar{B}_2), \bar{p}_1, \bar{a}, (\bar{B}'_1|\bar{A}|\bar{B}_2)) == \mathsf{true}, \\ &(\mathsf{exec}(\bar{B}_2, \bar{p}_2, \bar{a}, \bar{B}'_2) \wedge \neg\mathsf{member}(\bar{a}, \bar{A})) \supset \\ &\quad \mathsf{exec}((\bar{B}_1|\bar{A}|\bar{B}_2), \bar{p}_2, \bar{a}, (\bar{B}_1|\bar{A}|\bar{B}'_2)) == \mathsf{true}, \\ &(\mathsf{exec}(\bar{B}_1, \bar{p}_1, \bar{a}, \bar{B}'_1) \wedge \\ &\quad \mathsf{exec}(\bar{B}_2, \bar{p}_2, \bar{a}, \bar{B}'_2) \wedge \mathsf{member}(\bar{a}, \bar{A})) \supset \\ &\quad \mathsf{exec}((\bar{B}_1|\bar{A}|\bar{B}_2), \bar{p}_1 \wedge \bar{p}_2, \bar{a}, (\bar{B}'_1|\bar{A}|\bar{B}'_2)) == \mathsf{true}.\end{aligned}$$

- Conditional:

$$\begin{aligned}&\mathsf{exec}(\bar{B}, \bar{p}, \bar{a}, \bar{B}') \supset \\ &\quad \mathsf{exec}([\bar{p}'] \mathbin{{-}{>}} \bar{B}, \bar{p} \wedge \bar{p}', \bar{a}, \bar{B}') == \mathsf{true}.\end{aligned}$$

- Enabling:

$$\begin{aligned}&(\mathsf{exec}(\bar{B}_1, \bar{p}_1, \bar{a}, \bar{B}'_1) \wedge \neg\mathsf{member}(\bar{a}, \bar{A})) \supset \\ &\quad \mathsf{exec}((\bar{B}_1 \gg \bar{A} \gg \bar{B}_2), \bar{p}_1, \bar{a}, (\bar{B}'_1 \gg \bar{A} \gg \bar{B}_2)) == \mathsf{true}, \\ &(\mathsf{exec}(\bar{B}_1, \bar{p}_1, \bar{a}, \bar{B}'_1) \wedge \mathsf{member}(\bar{a}, \bar{A})) \supset \\ &\quad \mathsf{exec}((\bar{B}_1 \gg \bar{A} \gg \bar{B}_2), \bar{p}_1, \bar{a}, \bar{B}_2) == \mathsf{true}.\end{aligned}$$

- Disabling:

$$\begin{aligned}&(\mathsf{exec}(\bar{B}_1, \bar{p}_1, \bar{a}, \bar{B}'_1) \wedge \neg\mathsf{member}(\bar{a}, \bar{A})) \supset \\ &\quad \mathsf{exec}((\bar{B}_1 \,[\!>\, \bar{A} \,[\!>\, \bar{B}_2), \bar{p}_1, \bar{a}, (\bar{B}'_1 \,[\!>\, \bar{A} \,[\!>\, \bar{B}_2)) == \mathsf{true}, \\ &(\mathsf{exec}(\bar{B}_1, \bar{p}_1, \bar{a}, \bar{B}'_1) \wedge \mathsf{member}(\bar{a}, \bar{A})) \supset \\ &\quad \mathsf{exec}((\bar{B}_1 \,[\!>\, \bar{A} \,[\!>\, \bar{B}_2), \bar{p}_1, \bar{a}, \bar{B}'_1) == \mathsf{true}, \\ &\mathsf{exec}(\bar{B}_2, \bar{p}_2, \bar{a}, \bar{B}'_2) \supset \\ &\quad \mathsf{exec}((\bar{B}_1 \,[\!>\, \bar{A} \,[\!>\, \bar{B}_2), \bar{p}_2, \bar{a}, \bar{B}'_2) == \mathsf{true}.\end{aligned}$$

using relation $EXEC$. We introduce a production

$$\mathsf{Bool} \rightarrow \mathsf{bexp}(\mathsf{B\_exp}, \mathsf{State})$$

into P, where $\mathsf{bexp} \in T_{\mathrm{INT}}$, and the axioms shown in Table 5 into AX_{INT}.

5 Translation from Natural Language Specifications into BE Programs

5.1 Translation from Natural Language Specifications into Logical Formulae

In Refs. [3] and [6], we proposed a translation method from natural language specifications into algebraic specifications. As an example of an input specification, we considered a communication protocol specification. In such a specification, a sentence often specifies an action which a program (or a protocol machine in the case of protocol specifications) has to perform but often specifies implicitly when the action is performed.

Example 1: Consider the following consecutive sentences in Ref. [4]:

Table 4: Axioms on val.

$\bar{s}$: State

- Calculation: For each $c \in T_0$,

  $$\mathsf{val}(c, \bar{s}) == c,$$

  and for each $f \in T_0$ such that $A \to f(A_1, \ldots, A_n) \in P_0$,

  $\bar{t}_1 : A_1, \ \ldots, \ \bar{t}_n : A_n$

  $$\mathsf{val}(f(\bar{t}_1, \ldots, \bar{t}_n), \bar{s}) == f(\mathsf{val}(\bar{t}_1, \bar{s}), \ldots, \mathsf{val}(\bar{t}_n, \bar{s})).$$

- Initial value: For each $reg \in REG$ and $buf \in BUF$,

  $$\mathsf{val}(reg, \mathsf{s_{init}}) == \mathsf{initial}(reg),$$
  $$\mathsf{val}(buf, \mathsf{s_{init}}) == \mathsf{initial}(buf).$$

- $\mathsf{in}(buf, reg)$: For each $reg, reg' \in REG$ such that $reg \neq reg'$, and for each $buf, buf' \in BUF$ such that $buf \neq buf'$,

  $$\begin{aligned}\mathsf{val}(reg, \delta(\bar{s}, \mathsf{in}(buf, reg))) &== \mathsf{head}(\mathsf{val}(buf, \bar{s})),\\ \mathsf{val}(reg', \delta(\bar{s}, \mathsf{in}(buf, reg))) &== \mathsf{val}(reg', \bar{s}),\\ \mathsf{val}(buf, \delta(\bar{s}, \mathsf{in}(buf, reg))) &== \mathsf{tail}(\mathsf{val}(buf, \bar{s})),\\ \mathsf{val}(buf', \delta(\bar{s}, \mathsf{in}(buf, reg))) &== \mathsf{val}(buf', \bar{s}).\end{aligned}$$

- $\mathsf{out}(buf, reg)$: For each $reg, reg' \in REG$, and for each $buf, buf' \in BUF$ such that $buf \neq buf'$,

  $$\begin{aligned}\mathsf{val}(reg', \delta(\bar{s}, \mathsf{out}(buf, reg))) &== \mathsf{val}(reg', \bar{s}),\\ \mathsf{val}(buf, \delta(\bar{s}, \mathsf{out}(buf, reg))) &== \mathsf{val}(buf, \bar{s}) \cdot \mathsf{val}(reg, \bar{s}),\\ \mathsf{val}(buf', \delta(\bar{s}, \mathsf{out}(buf, reg))) &== \mathsf{val}(buf', \bar{s}).\end{aligned}$$

- $\mathsf{set}(reg \leftarrow t)$: For each $reg, reg' \in REG$ such that $reg \neq reg'$, and for each $buf' \in BUF$,

  $\bar{t} : type[reg]$

  $$\begin{aligned}\mathsf{val}(reg, \delta(\bar{s}, \mathsf{set}(reg \leftarrow \bar{t}))) &== \mathsf{val}(\bar{t}, \bar{s})\\ \mathsf{val}(reg', \delta(\bar{s}, \mathsf{set}(reg \leftarrow \bar{t}))) &== \mathsf{val}(reg', \bar{s}),\\ \mathsf{val}(buf', \delta(\bar{s}, \mathsf{set}(reg \leftarrow \bar{t}))) &== \mathsf{val}(buf', \bar{s}).\end{aligned}$$

Table 5: Axioms on bexp.

$\bar{\pi}$: B_id, $\bar{B}, \bar{B}'$: B_exp, $\bar{s}$: State,
$\bar{p}$: Bool, $\bar{a}$: Action

$$\mathsf{main}(\bar{\pi}) \supset \mathsf{bexp}(\bar{\pi}, \mathsf{s_{init}}) == \mathsf{true},$$
$$(\mathsf{bexp}(\bar{B}, \bar{s}) \wedge \mathsf{exec}(\bar{B}, \bar{p}, \bar{a}, \bar{B}') \wedge \mathsf{val}(\bar{p}, \bar{s})) \supset \mathsf{bexp}(\bar{B}', \delta(\bar{s}, \bar{a})) == \mathsf{true}.$$

(1) A valid incoming MAJOR SYNC POINT SPDU (with ...) results in an S-SYNC-MAJOR indication.
(2) If Vsc is false, V(A) is set equal to V(M).

The protocol machine has to perform the actions specified by (2) immediately after it performs the actions specified by (1). However, sentence (2) does not specify explicitly when the actions has to be performed. □

In Ref. [3], a state of the program which is specified implicitly in a natural language specification is called a situation. Moreover, for a constituent (i.e., a phrase, clause or sentence) X, the pre-situation of X is defined as the situation at which the action(s) specified by X has to be performed, and the post-situation of X is defined as the one immediately after the action(s) is performed. The pre-/post-situations are also defined for a sequence of sentences.

It is assumed that a natural language specification is a set of paragraphs (i.e., sequences of sentences) and there exists no contextual dependency between distinct paragraphs (i.e., for any constituent X in a paragraph, the pre-situation of X is either the pre-situation of the paragraph or the post-situation of another constituent). Each paragraph in a natural language specification is independently translated into an algebraic axiom.

In Ref. [3], we formalized situations as a data type Situation, which is represented by a sequence of "events" which the program has performed from the initial situation. By using type Situation and other primitive data types, a paragraph P of a natural language specification is translated into an axiom in the form of

$$\bar{x}_1 : A_1, \ldots, \bar{x}_m : A_m$$
$$Q_1\bar{x}_1 \cdots Q_m\bar{x}_m$$
$$\left[R_1 \ @ \left[\cdots \left[R_m \ @ \left[\bigwedge_{S \in P} pred_S\right]\right] \cdots\right]\right] == \mathsf{true}, \quad (1)$$

where $pred_S$ is a logical formula without quantifiers denoting the meaning of sentence S, $\bar{x}_1, \ldots, \bar{x}_m$ are all the distinct variables appearing in $\bigwedge_{S \in P} pred_S$, and A_j $(1 \leq j \leq m)$ is the data type of $\bar{x}_j$. We mean by $R_j \ @ \ F$ that $R_j(\bar{x}_1, \ldots, \bar{x}_j) \wedge F$ if $Q_j = \exists$, and $R_j(\bar{x}_1, \ldots, \bar{x}_j) \supset F$ if $Q_j = \forall$.

Each sub-predicate of $pred_S$ which denotes events has two extra parameters: One denotes the pre-situation and the other does the post-situation. In Axiom (1), each pre-/post-situation is represented by a variable of type Situation. The same situations are represented by the identical variable. Since each paragraph is "contextually closed," the variable representing the pre-situation of the paragraph is universally quantified, and the other variables of type Situation are existentially quantified. In the case of a protocol specification, the pre-situation of a paragraph usually denotes a state in which a protocol machine is waiting for an input. And, if a protocol machine reaches the post-situation of a paragraph, then the machine waits for a next input.

Example 2: The two sentences in Example 1 are translated into an axiom of $F ==$ true, where F is the following formula in a first-order predicate logic:

$$\bar{\sigma}_1, \ldots, \bar{\sigma}_8 : \mathsf{Situation}, \quad \bar{x}_1 : \mathsf{SPDU}, \quad \bar{x}_2 : \mathsf{SSprm}$$
$$\forall\bar{\sigma}_1\forall\bar{x}_1\exists\bar{\sigma}_2\exists\bar{x}_2\exists\bar{\sigma}_3\exists\bar{\sigma}_4\exists\bar{\sigma}_5\exists\bar{\sigma}_6\exists\bar{\sigma}_7\exists\bar{\sigma}_8$$
$$\begin{aligned}&[\mathsf{valid}(\bar{x}_1) \wedge \mathsf{incoming}(\bar{x}_1) \wedge \mathsf{MAP}(\bar{x}_1) \supset\\ &\quad [\mathsf{SSYNMind}(\bar{x}_2) \wedge\\ &\quad\quad [\mathsf{receive}(\bar{x}_1, \bar{\sigma}_1, \bar{\sigma}_2) \wedge \mathsf{send}(\bar{x}_2, \bar{\sigma}_2, \bar{\sigma}_3)] \wedge\\ &\quad\quad [\mathsf{if\_then\_else}(\mathsf{Vsc} = \mathsf{false},\\ &\quad\quad\quad\quad \mathsf{set\_equal\_to}(\mathsf{Va}, \mathsf{Vm}, \bar{\sigma}_4, \bar{\sigma}_5),\\ &\quad\quad\quad\quad \mathsf{nop}(\bar{\sigma}_6, \bar{\sigma}_7),\\ &\quad\quad\quad\quad \bar{\sigma}_3, \bar{\sigma}_8)]]].\end{aligned}$$

Intuitive meanings of subexpressions in the formula are presented in Table 6. □

Consider Axiom (1) again. Let $\bar{x}_j$ be an existentially quantified variable, and $\bar{x}_{j_1}, \ldots, \bar{x}_{j_l}$ be all the universally quantified variables such that $j_1, \ldots, j_l < j$. To simplify the implementation method proposed in the next section, we assume the existence of a Skolem function $skolem_{\bar{x}_j}$ for $\bar{x}_j$ such that if

$$R_k(x_1, \ldots, x_k) \equiv \mathsf{true}$$

for each k $(1 \leq k \leq j-1)$, then

$$R_j(x_1, \ldots, x_{j-1}, skolem_{\bar{x}_j}(x_{j_1}, \ldots, x_{j_l})) \equiv \mathsf{true}.$$

By introducing such Skolem functions, we can ignore the restriction R_j of every existentially quantified variable

Table 6: Meanings of predicates.

- valid($\bar{x}_1$): $\bar{x}_1$ has a valid data format.
- incoming($\bar{x}_1$): $\bar{x}_1$ is an incoming object.
- MAP($\bar{x}_1$): $\bar{x}_1$ is a data unit MAJOR SYNC POINT SPDU.
- SSYNMind($\bar{x}_2$): $\bar{x}_2$ is a service primitive S-SYNC-MAJOR indication.
- receive($\bar{x}_1, \bar{\sigma}_1, \bar{\sigma}_2$): At situation $\bar{\sigma}_1$, the event "receipt of $\bar{x}_1$" is allowed to occur and the situation immediately after the event is $\bar{\sigma}_2$.
- send($\bar{x}_2, \bar{\sigma}_2, \bar{\sigma}_3$): At situation $\bar{\sigma}_2$, the event "sending $\bar{x}_2$" has to occur and the situation immediately after the event is $\bar{\sigma}_3$.
- set_equal_to(Va, Vm, $\bar{\sigma}_4, \bar{\sigma}_5$): At situation $\bar{\sigma}_4$, the event "setting the value of V(A) equal to the value of V(M)" has to occur and the situation immediately after the event is $\bar{\sigma}_5$.
- nop($\bar{\sigma}_6, \bar{\sigma}_7$): At situation $\bar{\sigma}_6$, a "meaningless" event such as "setting the value of a register equal to the value of the register" has to occur and the situation immediately after the event is $\bar{\sigma}_7$.
- if_then_else($q, pred_1, pred_2, \bar{\sigma}_3, \bar{\sigma}_8$): At situation $\bar{\sigma}_3$, the events specified by $pred_1$ occurs if q is true, and the events specified by $pred_2$ occurs otherwise. The situation immediately after these events is $\bar{\sigma}_8$.

$\bar{x}_j$, and Axiom (1) can be transformed into the following axiom (the parameters of each R'_j are omitted):

$$\bar{x}'_1 : A'_1, \ldots, \bar{x}'_{m'} : A'_{m'}$$
$$\left[R'_1 \supset \left[\cdots \left[R'_{m'} \supset \left[\bigwedge_{S \in P} pred_S\right]\right] \cdots\right]\right] == \text{true},$$

that is,

$$\bar{x}'_1 : A'_1, \ldots, \bar{x}'_{m'} : A'_{m'}$$
$$\bigwedge_{S \in P} \left[(R'_1 \wedge \cdots \wedge R'_{m'}) \supset pred_S\right] == \text{true}.$$

5.2 Implementation of Logical Formulae by Behavior Definitions

In Ref. [3], an "event" is considered as an "atomic action" of a protocol machine (in the case of protocol specifications) such as transmitting data or updating a particular register (See Table 6). However, "atomic action" is informally used in Ref. [3] since protocol machines are not defined formally.

In Sect. 4, we defined a BE interpreter, which can be a formal model of protocol machines. Now we consider "atomic actions" as expressions of type Action, i.e., we assume that type Situation can be identified with type State introduced in this paper.

Let $SPEC_{\mathrm{NL}}$ be a natural language specification, i.e., a set of paragraphs. Let $SPEC_{\mathrm{IPL}} = (G_{\mathrm{IPL}}, AX_{\mathrm{IPL}})$, $G_{\mathrm{IPL}} = (N_{\mathrm{IPL}}, T_{\mathrm{IPL}}, P_{\mathrm{IPL}})$ be the algebraic specification derived from $SPEC_{\mathrm{NL}}$, where AX_{IPL} consists of:

- axioms on primitive data types; and
- axioms in the following form:

$$\bar{s}_1 : \text{Situation}, \bar{x}_1 : A_1, \ldots, \bar{x}_m : A_m$$
$$\bigwedge_{S \in P} \left[(R_1 \wedge \cdots \wedge R_m) \supset pred_S\right] == \text{true}, \quad (2)$$

where $P \in SPEC_{\mathrm{NL}}$ is a paragraph, and for each j ($1 \le j \le m$), A_j denotes a primitive data type and R_j denotes the restriction on $\bar{x}_1, \ldots, \bar{x}_j$. For any $P \in SPEC_{\mathrm{NL}}$, the pre-situation of P is denoted by $\bar{s}_1$, and for any $P_1, P_2 \in SPEC_{\mathrm{NL}}$, no variable except $\bar{s}_1$ appears in both P_1 and P_2.

$SPEC_{\mathrm{IPL}}$ is translated into a BE program $SPEC_{\mathrm{PRG}} = (G_{\mathrm{PRG}}, AX_{\mathrm{PRG}})$ in the following way. Let $SPEC_{\mathrm{INT}}$ be a BE interpreter for $SPEC_{\mathrm{PRG}}$, and $SPEC = SPEC_{\mathrm{PRG}} \cup SPEC_{\mathrm{INT}}$.

The idea of our implementation method is making each variable of a primitive data type correspond to a register and each variable of type Situation correspond to a behavior identifier. To do this, we introduce the following registers and behavior identifiers into $SPEC_{\mathrm{PRG}}$:

1. A set $REG_{\mathrm{VAR}} = \{var_1, \ldots, var_m\}$ of registers: var_j is used for storing the value of variable $\bar{x}_j$ in Axiom (2);
2. Let $REG_{\mathrm{PRED}} = \{reg_1, \ldots, reg_n\}$ be the set of registers which have been defined in $SPEC_{\mathrm{IPL}}$ (e.g., Vsc, Va, and Vm in Example 2). And, for each Skolem function $skolem_{\bar{x}}$ (introduced in Sect. 5.1) such that the type of $\bar{x}$ is not Situation, we suppose the existence of a function $\varphi_{\bar{x}}$ such that
$$skolem_{\bar{x}}(\bar{s}_1, \bar{x}_1, \ldots, \bar{x}_j) \equiv_{SPEC}$$
$$\varphi_{\bar{x}}(\text{val}(reg_1, \bar{s}_1), \ldots, \text{val}(reg_n, \bar{s}_1), \bar{x}_1, \ldots, \bar{x}_j).$$
Since a BE interpreter can calculate $\varphi_{\bar{x}}(\text{val}(reg_1, s), \ldots, \text{val}(reg_n, s), t_1, \ldots, t_j)$ only when it is in situation s, we introduce a set $REG_{\mathrm{BACK}} = \{back_1, \ldots, back_n\}$ of registers. The value of each $reg_i \in REG_{\mathrm{PRED}}$ ($1 \le i \le n$) is copied to $back_i \in REG_{\mathrm{BACK}}$ before the actions specified by a paragraph P are performed. Therefore, a BE interpreter can calculate $\varphi_{\bar{x}}(back_1, \ldots, back_n, t_1, \ldots, t_j)$ to obtain the value of $\varphi_{\bar{x}}(\text{val}(reg_1, s), \ldots, \text{val}(reg_n, s), t_1, \ldots, t_j)$. Similarly, we suppose that each restriction R_j is in the form of
$$R_j(\text{val}(reg_1, \bar{s}_1), \ldots, \text{val}(reg_n, \bar{s}_1), \bar{x}_1, \ldots, \bar{x}_j);$$
3. A set REG_{TMP} of temporary or dummy registers: An element of REG_{TMP} is denoted by tmp with some subscripts; and
4. For each pre-/post-situation $\hat{s}$ appearing in Axiom (2), we introduce a behavior identifier $\pi_{\hat{s}}$ into $SPEC_{\mathrm{PRG}}$. A human implementor specifies behavior definitions so that for any instantiated term s of $\hat{s}$ which satisfies Axiom (2),
   - bexp$(\pi_{\hat{s}}, s) \equiv_{SPEC}$ true, or
   - bexp$(\pi_{\hat{s}'}, s) \equiv_{SPEC}$ true for some $\pi_{\hat{s}'}$ such that $\pi_{\hat{s}'} := \pi_{\hat{s}} \equiv_{SPEC}$ true.

   That is, in such a situation s, a BE interpreter can always execute $\pi_{\hat{s}}$.

The implementation method consists of the following four steps:

Step 1: For each paragraph $P \in SPEC_{\mathrm{NL}}$, the set β_P of behavior definitions is constructed by the following three steps. $\bigcup_{P \in SPEC_{\mathrm{NL}}} \beta_P$ is the implementation of $SPEC_{\mathrm{NL}}$.

Step 2: For each sentence $S \in P$, $(R_1 \wedge \cdots \wedge R_m) \supset pred_S$ is translated into behavior definitions (denoted by $behavior[\langle R_1, \ldots, R_m\rangle, pred_S]$) as follows:

Step 2.1: The "subroutines" for "variable bindings" are defined as a set of behavior definitions (denoted by $bind[\langle R_1, \ldots, R_m\rangle, pred_S]$). Let $\alpha\{\gamma'_1/\gamma_1, \ldots, \gamma'_m/\gamma_m\}$

denote the expression obtained by replacing each subexpression γ_j $(1 \leq j \leq m)$ of expression α by expression γ'_j. Let $pre[pred_S]$ denote the actual parameter of $pred_S$ which represents the pre-situation of $pred_S$, and let $post[pred_S]$ denote the actual parameter of $pred_S$ which represents the post-situation of $pred_S$. Let $skolem_{\bar{s}}$ be the Skolem function for $\bar{s}$ introduced in Sect. 5.1. For simplicity, define $skolem_{\bar{s}_1}$, where $\bar{s}_1$ denotes the pre-situation of the paragraph, as the identity function on type Situation. Suppose that $pre[pred_S] = skolem_{\bar{s}}(\bar{s}_1, \bar{x}_1, \ldots, \bar{x}_j)$ $(0 \leq j \leq m)$ and $post[pred_S] = skolem_{\bar{s}'}(\bar{s}_1, \bar{x}_1, \ldots, \bar{x}_{j'})$ $(j \leq j' \leq m)$. Then, during the "execution" of $pred_S$, each of $\bar{x}_{j+1}, \ldots, \bar{x}_{j'}$ must be bound to some value. A behavior identifier which simulates these variable bindings is denoted by $\rho_{\hat{s},\hat{s}'}$. The behavior definition of $\rho_{\hat{s},\hat{s}'}$ is in the following form:

$$\begin{aligned}
\rho_{\hat{s},\hat{s}'} := {} & \mathsf{set}(var_{j+1} \leftarrow \hat{t}_{j+1}); \cdots ; \mathsf{set}(var_{j'} \leftarrow \hat{t}_{j'});\\
& ([(R_{j+1} \wedge \cdots \wedge R_{j'})\\
& \qquad \{var_1/\bar{x}_1, \ldots, var_{j'}/\bar{x}_{j'}\}] \text{->}\\
& \quad \mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}); \mathsf{stop}\\
& \quad \Box\\
& [\neg(R_{j+1} \wedge \cdots \wedge R_{j'})\\
& \qquad \{var_1/\bar{x}_1, \ldots, var_{j'}/\bar{x}_{j'}\}] \text{->}\\
& \quad \mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{false}); \mathsf{stop}).
\end{aligned}$$

Here, $\hat{t}_{j+1}, \ldots, \hat{t}_{j'}$ are terms which indicate how the values of $var_{j+1}, \ldots, var_{j'}$ are obtained respectively, and are specified by a human implementor. Register tmp_{bind} is used for storing the "return value" of $\rho_{\hat{s},\hat{s}'}$. Action $\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true})$ is performed if the variable bindings are completed successfully, and action $\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{false})$ is performed otherwise.

Step 2.2: The "main routine" of $pred_S$ is defined as a set of behavior definitions (denoted by $dic[pred_S]$). As shown in the following examples, a behavior definition of $\pi_{pre}[pred_S]$ has to be in $dic[pred_S]$, and $\pi_{post}[pred_S]$ has to appear in some behavior definitions in $dic[pred_S]$.

If $dic[pred_S]$ is already defined and stored as a "lexical item" of $pred_S$, then a human implementor has only to define $bind[\langle R_1, \ldots, R_m\rangle, pred_S]$.

Example 3: $dic[\mathsf{receive}(\hat{t}, \hat{s}, \hat{s}')]$ is shown in Table 7, where $type[tmp_{\mathrm{in}}] = D$ such that

$$\mathsf{Bool} \rightarrow \mathsf{receive}(D, \mathsf{Situation}, \mathsf{Situation}) \in P_{\mathrm{IPL}}.$$

The meaning of the behavior definition is as follows. When buf_1 is not empty, then look ahead the first element d of buf_1, copy d to a temporary register tmp_{in}, and perform variable bindings $\rho_{\hat{s},\hat{s}'}$. During the execution of $\rho_{\hat{s},\hat{s}'}$, $\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true})$ is performed if the variable bindings are completed successfully, and $\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{false})$ is performed otherwise. When the variable bindings are completed successfully, move the first element d of buf_1 to tmp_{in}, and execute $\pi_{\hat{s}'}$. Otherwise, execute $\pi_{\hat{s}}$, since the semantics of $\mathsf{receive}(\hat{t}, \hat{s}, \hat{s}')$ is "receipt of $\hat{t}$ is *allowed* to occur" (see Table 6) and the receipt of another data may be allowed by another axiom.

Also,

$$\begin{aligned}
& bind[\langle \mathsf{valid}(\bar{x}_1) \wedge \mathsf{incoming}(\bar{x}_1) \wedge \mathsf{MAP}(\bar{x}_1)\rangle,\\
& \qquad \mathsf{receive}(\bar{x}_1, \bar{\sigma}_1, \bar{\sigma}_2)]
\end{aligned}$$

Table 7: $dic[\mathsf{receive}(\hat{t}, \hat{s}, \hat{s}')]$.

$$\begin{aligned}
\pi_{\hat{s}} := {} & ([buf_1 \neq \lambda] \text{->} \mathsf{set}(tmp_{\mathrm{in}} \leftarrow \mathsf{head}(buf_1)); \rho_{\hat{s},\hat{s}'}\\
& \quad \gg \lambda\cdot\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true})\cdot\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{false}) \gg\\
& ([(tmp_{\mathrm{bind}} = \mathsf{true})\\
& \quad \wedge(tmp_{\mathrm{in}} = \hat{t}\{var_1/\bar{x}_1, \ldots, var_m/\bar{x}_m\})] \text{->}\\
& \qquad \mathsf{in}(buf_1, tmp_{\mathrm{in}}); \pi_{\hat{s}'}\\
& \quad \Box\\
& [\neg((tmp_{\mathrm{bind}} = \mathsf{true})\\
& \quad \wedge(tmp_{\mathrm{in}} = \hat{t}\{var_1/\bar{x}_1, \ldots, var_m/\bar{x}_m\}))] \text{->} \pi_{\hat{s}}))
\end{aligned}$$

Table 8: $dic[\mathsf{send}(\hat{t}, \hat{s}, \hat{s}')]$.

$$\begin{aligned}
\pi_{\hat{s}} := {} & (\rho_{\hat{s},\hat{s}'}\\
& \qquad \gg \lambda\cdot\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}) \gg\\
& \quad \mathsf{set}(tmp_{\mathrm{out}} \leftarrow \hat{t}\{var_1/\bar{x}_1, \ldots, var_m/\bar{x}_m\});\\
& \quad \mathsf{out}(buf_2, tmp_{\mathrm{out}}); \pi_{\hat{s}'})
\end{aligned}$$

can be defined as

$$\begin{aligned}
\rho_{\bar{\sigma}_1,\bar{\sigma}_2} := {} & \mathsf{set}(var_1 \leftarrow tmp_{\mathrm{in}});\\
& ([\mathsf{valid}(var_1) \wedge \mathsf{incoming}(var_1)\\
& \qquad \wedge\mathsf{MAP}(var_1)] \text{->}\\
& \quad \mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}); \mathsf{stop}\\
& \quad \Box\\
& [\neg(\mathsf{valid}(var_1) \wedge \mathsf{incoming}(var_1)\\
& \qquad \wedge\mathsf{MAP}(var_1))] \text{->}\\
& \quad \mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{false}); \mathsf{stop}).
\end{aligned}$$

Here, we specify that the value of $\bar{x}_1$ is equal to the value of tmp_{in}, i.e., the first element of buf_1. □

Example 4: We define $dic[\mathsf{send}(\hat{t}, \hat{s}, \hat{s}')]$ as shown in Table 8, where $type[tmp_{\mathrm{out}}] = D$ such that

$$\mathsf{Bool} \rightarrow \mathsf{send}(D, \mathsf{Situation}, \mathsf{Situation}) \in P_{\mathrm{IPL}}.$$

First, perform the variable bindings $\rho_{\hat{s},\hat{s}'}$. When this is completed successfully, calculate $\hat{t}$, assign the result to a temporary register tmp_{out}, and output it to buf_2. Otherwise, come to a deadlock, since the semantics of $\mathsf{send}(\hat{t}, \hat{s}, \hat{s}')$ is "sending $\hat{t}$ *has to* occur" (see Table 6) and the variable bindings also *have to* be completed successfully.

The behavior definition

$$\begin{aligned}
& bind[\langle \mathsf{valid}(\bar{x}_1) \wedge \mathsf{incoming}(\bar{x}_1) \wedge \mathsf{MAP}(\bar{x}_1)\rangle,\\
& \qquad \mathsf{send}(\bar{x}_2, \bar{\sigma}_2, \bar{\sigma}_3)]
\end{aligned}$$

is simply defined as $\rho_{\bar{\sigma}_2,\bar{\sigma}_3} := \mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}); \mathsf{stop}$, since there are no variables to be bound. □

Example 5: We define $dic[\mathsf{set\_equal\_to}(\hat{t}_1, \hat{t}_2, \hat{s}, \hat{s}')]$ as shown in Table 9. Also,

$$\begin{aligned}
& bind[\langle \mathsf{valid}(\bar{x}_1) \wedge \mathsf{incoming}(\bar{x}_1) \wedge \mathsf{MAP}(\bar{x}_1)\rangle,\\
& \qquad \mathsf{set\_equal\_to}(\mathsf{Va}, \mathsf{Vm}, \bar{\sigma}_4, \bar{\sigma}_5)]
\end{aligned}$$

can be defined as $\rho_{\bar{\sigma}_4,\bar{\sigma}_5} := \mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}); \mathsf{stop}$. □

As shown in the following example, we can construct behavior definitions for a predicate which takes other predicates as its parameters.

Example 6: We define

$$behavior[\langle R_1, \ldots, R_m\rangle, \mathsf{if\_then\_else}(\hat{q}, \hat{p}_1, \hat{p}_2, \hat{s}, \hat{s}')]$$

Table 9: $dic[\mathsf{set\_equal\_to}(\hat{t}_1, \hat{t}_2, \hat{s}, \hat{s}')]$

$$\begin{aligned}
\pi_{\hat{s}} \;:=\; (&\rho_{\hat{s},\hat{s}'} \\
&\quad \gg \lambda\cdot\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}) \gg \\
&\mathsf{set}(\hat{t}_1 \leftarrow \hat{t}_2\{var_1/\bar{x}_1, \ldots, var_m/\bar{x}_m\}); \pi_{\hat{s}'})
\end{aligned}$$

as

$$\begin{aligned}
&behavior[\langle R_1, \ldots, R_m\rangle, \hat{p}_1],\\
&behavior[\langle R_1, \ldots, R_m\rangle, \hat{p}_2],\\
&dic[\mathsf{if\_then\_else}(\hat{q}, \hat{p}_1, \hat{p}_2, \hat{s}, \hat{s}')].
\end{aligned}$$

And, $dic[\mathsf{if\_then\_else}(\hat{q}, \hat{p}_1, \hat{p}_2, \hat{s}, \hat{s}')]$ is the set of the following behavior definitions:

$$\begin{aligned}
\pi_{\hat{s}} &:= ([\hat{q}\{var_1/\bar{x}_1, \ldots, var_m/\bar{x}_m\}] \mathrel{{-}{>}} \pi_{pre[\hat{p}_1]}\\
&\qquad \square\\
&\qquad [\neg\hat{q}\{var_1/\bar{x}_1, \ldots, var_m/\bar{x}_m\}] \mathrel{{-}{>}} \pi_{pre[\hat{p}_2]}),\\
\pi_{post[\hat{p}_1]} &:= \pi_{\hat{s}'},\\
\pi_{post[\hat{p}_2]} &:= \pi_{\hat{s}'}.
\end{aligned}$$

□

Let $\beta'_P = \bigcup_{S\in P} behavior[\langle R_1, \ldots, R_m\rangle, pred_S]$.

Step 3: The following behavior definition is added to β'_P:

$$\begin{aligned}
\rho_{\mathrm{back}} := \;&\mathsf{set}(back_1 \leftarrow reg_1); \cdots; \mathsf{set}(back_n \leftarrow reg_n);\\
&\mathsf{set}(tmp_{\mathrm{back}} \leftarrow tmp_{\mathrm{back}});\\
&\mathsf{stop}.
\end{aligned}$$

Then, each $\pi_{\bar{s}_1} := B$ in β'_P is replaced by

$$\pi_{\bar{s}_1} := (\rho_{\mathrm{back}} \gg \lambda \cdot \mathsf{set}(tmp_{\mathrm{back}} \leftarrow tmp_{\mathrm{back}}) \gg B).$$

ρ_{back} is executed first of all in order to remember the value of each register in REG_{PRED} at the pre-situation of P. Action $\mathsf{set}(tmp_{\mathrm{back}} \leftarrow tmp_{\mathrm{back}})$ is used as a "signal" which denotes the completion of copying reg_i to $back_i$ ($1 \leq i \leq n$). Let β''_P be the resultant set of behavior definitions.

Step 4: Let $\bar{s}$ be the post-situation of paragraph P. Then, $\pi_{\bar{s}} := \pi_{\bar{s}_1}$ is added to β''_P. That is, after the completion of performing the actions specified by P, a BE interpreter executes $\pi_{\bar{s}_1}$, i.e., performs the actions specified by any paragraph P' from the first (recall that the pre-situation of any paragraph P' is denoted by $\bar{s}_1$). The resultant set of behavior definitions is β_P.

Example 7: For the logical formula in Example 2, the behavior definitions in Table 10 are obtained. Here, we write π_i instead of $\pi_{\bar{\sigma}_i}$. By Step 3, the behavior definition of π_1 is modified so that ρ_{back} is executed first. And, by Step 4, the behavior definition of π_8 is added. □

Table 10: Implementation of the logical formula in Example 2.

$$\begin{aligned}
\pi_1 \;:=\; (&\rho_{\mathrm{back}}\\
&\quad \gg \lambda\cdot\mathsf{set}(tmp_{\mathrm{back}} \leftarrow tmp_{\mathrm{back}}) \gg\\
&([buf_1 \neq \lambda] \mathrel{{-}{>}} \mathsf{set}(tmp_{\mathrm{in}} \leftarrow \mathsf{head}(buf_1)); \rho_{\bar{\sigma}_1,\bar{\sigma}_2}\\
&\quad \gg \lambda\cdot\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true})\cdot\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{false}) \gg\\
&([(tmp_{\mathrm{bind}} = \mathsf{true}) \wedge (tmp_{\mathrm{in}} = var_1)] \mathrel{{-}{>}}\\
&\qquad \mathsf{in}(buf_1, tmp_{\mathrm{in}}); \pi_2\\
&\quad \square\\
&[\neg((tmp_{\mathrm{bind}} = \mathsf{false}) \wedge (tmp_{\mathrm{in}} = var_1))] \mathrel{{-}{>}} \pi_1))),\\
\pi_2 \;:=\; (&\rho_{\bar{\sigma}_2,\bar{\sigma}_3}\\
&\quad \gg \lambda\cdot\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}) \gg\\
&\mathsf{set}(tmp_{\mathrm{out}} \leftarrow \varphi_{\bar{x}_2}(back_1, \ldots, back_n, var_1));\\
&\mathsf{out}(buf_2, tmp_{\mathrm{out}}); \pi_3),\\
\pi_3 \;:=\; (&[\mathsf{Vsc} = \mathsf{false}] \mathrel{{-}{>}} \pi_4 \;\square\; [\neg(\mathsf{Vsc} = \mathsf{false})] \mathrel{{-}{>}} \pi_6),\\
\pi_4 \;:=\; (&\rho_{\bar{\sigma}_4,\bar{\sigma}_5}\\
&\quad \gg \lambda\cdot\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}) \gg\\
&\mathsf{set}(\mathsf{Va} \leftarrow \mathsf{Vm}); \pi_5),\\
\pi_5 \;:=\; &\pi_8,\\
\pi_6 \;:=\; &\mathsf{set}(tmp_{\mathrm{nop}} \leftarrow tmp_{\mathrm{nop}}); \pi_7,\\
\pi_7 \;:=\; &\pi_8,\\
\pi_8 \;:=\; &\pi_1,
\end{aligned}$$

$$\begin{aligned}
\rho_{\mathrm{back}} \;:=\; &\mathsf{set}(back_1 \leftarrow reg_1); \cdots; \mathsf{set}(back_n \leftarrow reg_n);\\
&\mathsf{set}(tmp_{\mathrm{back}} \leftarrow tmp_{\mathrm{back}});\\
&\mathsf{stop},\\
\rho_{\bar{\sigma}_1,\bar{\sigma}_2} \;:=\; &\mathsf{set}(var_1 \leftarrow tmp_{\mathrm{in}});\\
&([\mathsf{valid}(var_1) \wedge \mathsf{incoming}(var_1)\\
&\qquad \wedge\mathsf{MAP}(var_1)] \mathrel{{-}{>}}\\
&\quad \mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}); \mathsf{stop}\\
&\qquad \square\\
&[\neg(\mathsf{valid}(var_1) \wedge \mathsf{incoming}(var_1)\\
&\qquad \wedge\mathsf{MAP}(var_1))] \mathrel{{-}{>}}\\
&\quad \mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{false}); \mathsf{stop}),\\
\rho_{\bar{\sigma}_2,\bar{\sigma}_3} \;:=\; &\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}); \mathsf{stop},\\
\rho_{\bar{\sigma}_4,\bar{\sigma}_5} \;:=\; &\mathsf{set}(tmp_{\mathrm{bind}} \leftarrow \mathsf{true}); \mathsf{stop}.
\end{aligned}$$

6 Conclusion

According to the definition of a BE interpreter, we have implemented a prototype system which executes a given BE program. Now we are implementing a system which translates algebraic specifications derived from natural language specifications into BE programs.

References

[1] Futatsugi, K. and Toyama, Y., "Term Rewriting Systems and Their Applications: A Survey," *Journal of IPSJ*, vol. 24, no. 2, pp. 133–146, Feb. 1983 (in Japanese).

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

[3] 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.

[4] ISO: "Basic Connection Oriented Session Protocol Specification," ISO 8327.

[5] 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, Jul. 1986 (in Japanese), also in *Systems and Computers in Japan*, vol. 18, no. 7, pp. 11–20, Jul. 1987.

[6] 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).