ソフトウェア工学 96－10
（1994. 1. 21）

自然語仕様から代数的仕様への変換における表現式の構文規則の生成

大崎 敦司[†] 石原 靖哲[†] 関 浩之[†‡] 嵩 忠雄[†‡]

† 大阪大学 基礎工学部 情報工学科
‡ 奈良先端科学技術大学院大学 情報科学研究科

筆者らは，自然語で記述されたプログラム仕様（自然語仕様と呼ぶ）から代数的仕様への変換法の研究を行っている．この変換法では，自然語仕様から代数的公理への変換を行うのみであり，公理で用いられるデータタイプの宣言や表現式の構文があらかじめ辞書項目として与えられていることを前提としている．本稿で述べる手法では，入力となる自然語仕様を，自然語の構文情報（文型などを指定した構文規則と，品詞・人称などを指定した辞書）をもとに解析することにより，公理に現れる表現式の構文規則を機械的に生成する．さらに，適切な部分データタイプの導入や，構文規則の簡単化を，変換システムのユーザと対話的に行なう．また，本稿では，この生成法に基づいて作成したシステムに，OSI セションプロトコル仕様中の 48 文を入力として与えた時の結果についても述べる．

Generation of Signatures of Functions in Translation from Natural Language Specifications into Algebraic Specifications

Atsushi Ohsaki[†] Yasunori Ishihara[†] Hiroyuki Seki[†‡] Tadao Kasami[†‡]

† Faculty of Engineering Science, Osaka University
‡ Nara Institute of Science and Technology

We have proposed a translation method from natural language specifications into algebraic axioms. However, this translation method assumed that the declarations of data types and the signatures of functions which appear in the algebraic specification to be derived by translation are predefined in the lexicon used for translation. In this paper, we propose a method of generating the signatures of functions by analyzing syntactic information of a natural language specification. The user of the translation system can add suitable subtypes to the generated signatures interactively. Moreover, we propose a procedure which simplifies the signatures based on the notion of structural equivalence of cfg's. We also show the result of applying the prototype system based on this method to a part of the OSI session protocol specification (48 sentences).

1 まえがき

文献 [1], [7] で筆者らは, 自然語仕様から代数的仕様への変換法を提案し, 変換システムを試作している. 代数的仕様 $SPEC$ とは, 2 節で述べるように, 始記号を指定しない文脈自由文法 G と, 公理の集合 AX との 2 字組 $SPEC = (G, AX)$ で表される. 文法 G によってデータタイプの宣言や表現式の構文 (関数名および引数と関数値のデータタイプの宣言) を指定する. 文法 G で導入された関数の意味は, 公理集合 AX 中のすべての公理を満たす最小の合同関係により形式的に定義される. 文献 [1], [7] で提案した変換法は, 自然語仕様から代数的公理 AX への変換を行なうのみであり, 文法 G はあらかじめ辞書項目として与えられていることを前提としている.

本稿で述べる手法では, 入力となる自然語仕様を, 自然語の構文情報をもとに解析することにより, 公理に現れる表現式の構文規則を機械的に生成する. さらに, 適切な部分データタイプの導入や, 構文規則の簡単化を, 変換者 (ユーザ) と対話的に行なう. また, この手法に基づく生成支援システムについても述べる.

2 代数的言語 ASL/*

本稿では, 代数的仕様の記述言語として ASL/* [6],[8] を採用する. ASL/* では, 2 字組 $SPEC = (G, AX)$ を代数的仕様と呼ぶ. ここで, G は始記号を指定しない文脈自由文法, AX は公理の集合である.

$G = (N, T, P)$ とし, N, T, P をそれぞれ非終端記号の集合, 終端記号の集合, 構文規則の集合とする. G において, $A \in N$ から 1 回の導出で $\alpha \in (N \cup T)^*$ が生成されるとき, $A \underset{G}{\Rightarrow} \alpha$ と書く. 関係 $\underset{G}{\Rightarrow}$ の反射推移閉包を $\overset{*}{\underset{G}{\Rightarrow}}$ で表す. $L_G[A] = \{w \in T^* \mid A \overset{*}{\underset{G}{\Rightarrow}} w\}$ とし, $L_G = \bigcup_{A \in N} L_G[A]$ とする. L_G の元を $SPEC$ の表現式と呼ぶ. 各非終端記号はデータタイプ (ソート) に対応しており, $exp \in L_G[A]$ のとき, exp のデータタイプ (あるいは, 型) は A であると言う. また, A, $A' \in N$ に対し, $A' \overset{*}{\underset{G}{\Rightarrow}} A$ ならば, A は A' の部分データタイプであると言う. 各終端記号は, 関数 (定数) 記号に対応している. 各構文規則は, 関数の構文宣言 (関数名および引数と関数値のデータタイプの宣言), あるいはデータタイプ間の包含関係 (以下, 型包含関係と呼ぶ) の宣言に対応している.

公理は変数を含む表現式の対 $exp == exp'$ の形で記述する. ただし, 公理に現れる各変数 $\bar{x}$ には, 非終端記号 $A_{\bar{x}} \in N$ が一つ対応づけられ, 変数 $\bar{x}$ には, $L_G[A_{\bar{x}}]$ の任意の元が代入できると考える. この $A_{\bar{x}}$ を $\bar{x}$ のタイプ指定と呼ぶ. AX のすべての公理を満たす L_G 上の最小の合同関係を代数的仕様 $SPEC$ の指定する合同関係と言う. 実際には, 代数的仕様によって規定しようとする系や世界において, 「表現式 exp と exp' が同一の対象 (例えば, データや状態遷移) を表すとき, exp と exp' が合同となる」ように仕様の記述を行う.

3 従来の変換法の概要と問題点

3.1 従来の変換法の概要

文献 [1], [7] で筆者らが提案した変換法では, 入力となる自然語仕様の各文を, 自然語の構文規則を用いて構文解析する. 次に, 各単語に対して定義されている中間表現 (後述) をもとに, 構文木にしたがって文に対する中間表現をボトムアップに構成する. さらに, 文脈の解析として, (a) 論理変数の量記号の決定, (b) 前方照応の解析, (c) 文章間の依存関係の解析を行う. 最後に, 1 階述語論理式の形の代数的公理を生成する. この変換法に基づく変換システムの概要を図 1に示す.

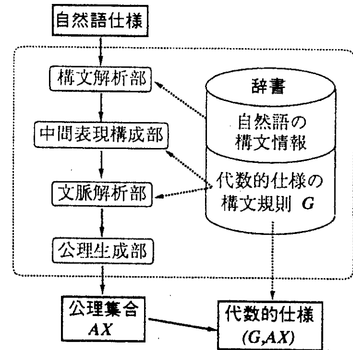

図 1: 変換システムの概要

表 1: 例 1の公理に現れる述語の意味

valid($\bar{x}_1$)	$\bar{x}_1$ は正常なデータフォーマットをもつ
incoming($\bar{x}_1$)	$\bar{x}_1$ は受信されるデータである
MAP($\bar{x}_1$)	$\bar{x}_1$ はデータ単位 MAJOR SYNC POINT SPDU である
SSYNMind($\bar{x}_2$)	$\bar{x}_2$ はサービスプリミティブ S-SYNC-MAJOR indication である

例 1: 例として, 文献 [2] 中の以下のような自然語文を考える.

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

この自然語文は, 次のような代数的公理に変換される.

$$\bar{x}_1 : \mathrm{In_SPDU} \quad \bar{x}_2 : \mathrm{Out_SSprm}$$
$$\forall \bar{x}_1 \exists \bar{x}_2 (\mathrm{valid}(\bar{x}_1) \wedge \mathrm{incoming}(\bar{x}_1) \wedge \mathrm{MAP}(\bar{x}_1) \supset (\mathrm{SSYNMind}(\bar{x}_2) \wedge \mathrm{result_in}(\bar{x}_1, \bar{x}_2))) == \mathrm{true}$$

ここで, $\bar{x}_1 : \mathrm{In_SPDU}$ は, $\bar{x}$ のタイプ指定が In_SPDU であることを宣言している. 上の公理に現れている各述語の意味は表 1の通りである (形式的には公理を用いて定義する). □

3.2 中間表現の構成

中間表現とは, 代数的公理を生成するのに必要な情報を表現する形式である. まず, C 構造を定義する. F を属性名の有限集合とする. F の各属性 f に対して, f のとり得る値の集合 $V(f)$ が定まっているとする. $V(f)$ の元は, 整数値, 論理値などの基本値, あるいは, C 構造自身, C 構造の集合や系列でもよい. 属性とその値の対の有限集合で, 属性名が等しく値の異なるような対を含まない任意の集合を C 構造と呼ぶ. C 構造 $\{\langle f_1, v_1\rangle, \ldots, \langle f_n, v_n\rangle\}$ を

$$\begin{bmatrix} f_1 & v_1 \\ \vdots & \vdots \\ f_n & v_n \end{bmatrix}$$

表 2: 本稿で用いる属性

trans	自然語文または句の意味を表す表現式
type	属性 trans で指定された表現式の代数的仕様におけるデータタイプ
subcat	属性 trans で指定される表現式が関数記号 (または述語記号) のとき, その引数に関する情報
restriction	属性 trans で指定される表現式が変数のとき, その変数の満たすべき前提条件

表 3: 以下の例で用いるデータタイプ

In_event	プロトコル機械に与えられる事象
Out_event	プロトコル機械が起こす事象
In_data	受信されるデータ
Out_data	送信されるデータ
SPDU	セションプロトコルデータ単位
SSprm	セションサービスプリミティブ
In_SPDU	受信されるセションプロトコルデータ単位
In_SSprm	受信されるセションサービスプリミティブ
Out_SPDU	送信されるセションプロトコルデータ単位
Out_SSprm	送信されるセションサービスプリミティブ

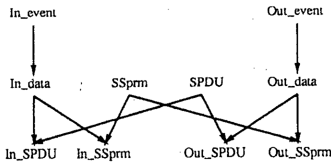

図 2: 表 3のデータタイプ間の包含関係

とも書く. 自然語の文や句に対する構文木の頂点 v に C 構造 C がラベル付けされているとき, C を v に付随する C 構造という. 各頂点に C 構造が付随しているような, 自然語の文や句に対する構文木を, 中間表現と呼ぶ.

本稿で用いる属性とその意味を表 2に示す. 自然語の単語に対する中間表現では, これらの属性を用いて, (a) 公理に現れる関数の構文宣言, (b) 公理に現れる変数のタイプ指定, (c) 公理に現れる変数の満たすべき前提条件の指定を行う.

例 2: 以下の例で用いるデータタイプとその意味を表 3に示す. また, それらの間の包含関係を図 2に示す. A から A' への有向枝は, $A \overset{*}{\underset{G}{\Rightarrow}} A'$を意味する. □

例 3: 図 3に, "in" を伴う "results" の中間表現の根頂点に付随する C 構造を示す. X_1, X_2 は, 中間表現中の変数 (指標) であり, 文や句に対する中間表現を構成していったとき, 同一の変数で表されている部分には同一の値が入ることを示している. この C 構造では,

$$\text{Bool} \rightarrow \text{result_in}(\text{In_event}, \text{Out_event})$$

に相当する構文の宣言がなされている. また, 図 4に, "S-SYNC-MAJOR indication" の中間表現の根頂点に付随する C 構造を示す. この C 構造では, "S-SYNC-MAJOR indication" は公理において変数で表され (属性 trans), タイプ指定は SSprm (の部分データタイプ) となること (属性 type), さらに, その変数は述語 SSYNMind を満たすこと (属性 restriction) を指定している. □

$$\begin{bmatrix} \text{trans} & \text{result_in}(X_1, X_2) \\ \text{type} & \text{Bool} \\ \text{subcat} & \begin{bmatrix} \begin{bmatrix} \text{trans} & X_2 \\ \text{type} & \text{Out_event} \end{bmatrix} \\ \begin{bmatrix} \text{trans} & X_1 \\ \text{type} & \text{In_event} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

図 3: "in" を伴う "results" の中間表現

$$\begin{bmatrix} \text{trans} & \bar{x} \\ \text{type} & \text{SSprm} \\ \text{restriction} & \begin{bmatrix} \begin{bmatrix} \text{trans} & \text{SSYNMind}(\bar{x}) \\ \text{type} & \text{Bool} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

図 4: "S-SYNC-MAJOR indication" の中間表現

$$\begin{bmatrix} \text{trans} & \text{result_in}(X_1, \bar{x}_2) \\ \text{type} & \text{Bool} \\ \text{subcat} & \begin{bmatrix} \begin{bmatrix} \text{trans} & \bar{x}_2 \\ \text{type} & \text{Out_SSprm} \\ \text{restriction} & \begin{bmatrix} \begin{bmatrix} \text{trans} & \text{SSYNMind}(\bar{x}_2) \\ \text{type} & \text{Bool} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \text{trans} & X_1 \\ \text{type} & \text{In_event} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

図 5: "results in an S-SYNC-MAJOR indication"

次に, 自然語の文や句に対する中間表現は, 部分句に対する中間表現と, そこで適用されている自然語の構文規則をもとに構成される. 自然語の各構文規則 $Y \rightarrow Y_1 \cdots Y_n$ に対し, 「右辺の非終端記号 Y_i から生成される語句が, 右辺のその他の非終端記号 $Y_j \neq Y_i$ から生成される語句を引数にとる関数に対応する」 ような Y_i が指定されている. 例えば,

$$\text{VP} \rightarrow \text{VT NP}$$

という自然語の構文規則 (VP, VT, NP はそれぞれ動詞句, 他動詞, 名詞句を生成する非終端記号) に対し, 「VT から生成される語句が, NP から生成される語句を引数にとる関数に対応する」 ことが指定されている. 構文木において, 関数に対応する非終端記号 Y_i がラベルづけされた頂点に付随する C 構造の属性 subcat の各値と, 他の非終端記号 $Y_j \neq Y_i$ に対応する頂点に付随する C 構造とに対して単一化 (2 つの C 構造のもつ情報を合わせもつ C 構造を作る操作) を行う. 特に, 属性 type の値の単一化は, 「2 つのデータタイプの部分データタイプであり, かつ, 最大のデータタイプ」 として定義される. 型包含関係 (2節参照) は中間表現を構成するための規則の一部として与えておく.

例 4: "results in an S-SYNC-MAJOR indication" に対する中間表現の根頂点に付随する C 構造は, 図 3の属性 subcat の 1 番目の値である C 構造と図 4の C 構造とを単一化することにより, 図 5のようになる. □

例 5: "valid", "incoming", および "MAJOR SYNC POINT SPDU" の中間表現の根頂点に付随する C 構造を, それぞれ図 6, 7, 8に示す. 図 6, 7の属性 subcat の値の C 構造と, 図 8の C 構造とを単一化し, さらに, この自然語の構文規則に対して定められている, C 構造に対する変形操作 (詳細は文献 [7]) を施すことにより, "a valid incoming MAJOR SYNC POINT SPDU" に対する中間表現の根頂点に付随する C 構造は図 9のようになる. □

このようにして得られた各文の中間表現に対して文脈の解析[1],[7]を行い, 例 1で示したような 1 階述語論理式の形の代数的公理を生成する.

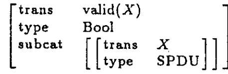

図 6: "valid" の中間表現

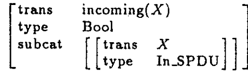

図 7: "incoming" の中間表現

図 8: "MAJOR SYNC POINT SPDU" の中間表現

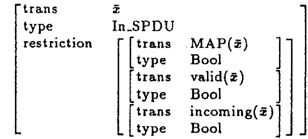

図 9: "a valid incoming MAJOR SYNC POINT SPDU" の中間表現

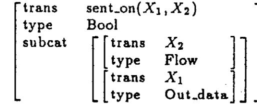

図 10: "sent" の中間表現

3.3 属性 type の導入の利点

適切なデータタイプを導入し, 適切な関数の構文宣言をすることによって, もとの自然語仕様でのあいまいさを減少できることがある.

例 6: 文献 [2] 中の次のような連続する 2 文を考える.

1. A valid incomming ABORT SPDU results in sending an ABORT ACCEPT SPDU.
2. This SPDU is sent on the transport normal flow.

"ABORT SPDU" と "ABORT ACCEPT SPDU" に対応する表現式 (この場合, 公理の変数) のデータタイプが SPDU 型であることだけがわかっているとすると, "this SPDU" がどちらの語句を照応しているのか決定できない.

ここで, 例2のようにデータタイプを導入し, L[In_event]∩L[Out_event] = ∅ であるとする. そして, 以下のように, 語句に対応する関数の構文宣言を行う.

- "result in" の主語, 目的語に対応する表現式のデータタイプは, それぞれ, In_event 型, Out_event 型である (図 3).
- "be sent on" の主語に対応する表現式のデータタイプは Out_data 型である (図 10).

これと図 2を用いて中間表現を構成すると,

- "ABORT SPDU" に対応する表現式 (公理の変数) のデータタイプは In_SPDU 型
- "ABORT ACCEPT SPDU" に対応する表現式 (公理の変数) のデータタイプは Out_SPDU 型
- "this SPDU" に対応する表現式 (公理の変数) のデータタイプは Out_SPDU 型

となる. "this SPDU" が "ABORT SPDU" を照応していると仮定すると, それぞれに対応する公理の変数は同一のものとなり, そのタイプ指定は単一化の定義を用いて「In_SPDU および Out_SPDU の部分データタイプのうち最大のもの」となる. 一方, L[In_event] ∩ L[Out_event] = ∅ であることから, L[In_SPDU] ∩ L[Out_SPDU] = ∅ であり, そのようなデータタイプは存在しない. したがって, "this SPDU" は "ABORT SPDU" を照応し得ないと決定できる. □

3.4 従来の変換法の問題点

文献 [1] や [7] で述べた変換法では, 自然語の各単語に対する中間表現は辞書においてあらかじめ定義されているという仮定をおいている. すなわち, 公理に現れる関数の構文宣言があらかじめ与えられているということを前提としている (図 3の属性 type, 属性 subcat 中の属性 type). 同様に, 属性 type の値の単一化を求めるのに必要な型包含関係についても, あらかじめ与えられているということを前提としている. しかし, データタイプを適切に導入・階層化し, 関数の構文を記述することは労力を必要とする作業である. 次節では, 入力となる自然語仕様を機械的に解析することによって関数の構文宣言の記述を支援する手法について述べる.

4 構文規則の生成法

代数的仕様 $SPEC = (G, AX)$ の G を生成するにあたり, 以下の仮定をおく.

- 自然語の構文情報 (文型などを指定した構文規則と, 品詞・人称などを指定した辞書) が与えられている.
- 自然語の語句に対する中間表現は与えられていない.
- 定義済みであることを前提としている, プリミティブなデータタイプ (Bool など) と, プリミティブなデータタイプ上の定数および演算の構文を定義する文脈自由文法 $G_0 = (N_0, T_0, P_0)$ が与えられている.

上の仮定のもとで, 本節で述べる手法では, 公理に現れる関数の構文宣言を機械的に生成する. さらに, 必要ならば, 変換システムのユーザ (以下, 変換者と呼ぶ) と対話的に非終端記号や構文規則を追加し, G を生成する. L_{AX} を, AX に属する公理の両辺に現れる表現式およびその部分式全体からなる有限集合とすると, 生成される G が以下の条件を満たしていることが望まれる.

(a) $L_{AX} \subseteq L_G$ を満たす.

(b) プリミティブなデータタイプ (Bool など) や, 変換者が導入したデータタイプ (4.4節参照) を表す非終端記号が N に含まれる.

(c) (a),(b) を満たす範囲で, "できるだけ簡単" である.

これらを満たすように G を以下の手順で生成する.

(1) 自然語仕様を自然語の構文情報をもとに解析し, L_{AX} を構成する.

$$\begin{bmatrix} \text{trans} & \text{result_in}(X_1, X_2) \\ \text{subcat} & \left[\begin{bmatrix}\text{trans} & X_2\end{bmatrix} \begin{bmatrix}\text{trans} & X_1\end{bmatrix}\right] \end{bmatrix} \quad (1)$$

$$\begin{bmatrix} \text{trans} & \bar{x} \\ \text{restriction} & \left[\begin{bmatrix}\text{trans} & \text{SSYNMind}(\bar{x})\end{bmatrix}\right] \end{bmatrix} \quad (2)$$

図 11: 機械的に構成された中間表現

(2) L_{AX} において, "どの表現式が, どの関数の何番目の引数に現れるか" の情報から, 型包含関係を解析し, それらの関係を満たすように G を構成する.

(3) 得られた G を構造的等価性に基づく最簡形に変換する.

(4) 必要ならば, 変換者が非終端記号や構文規則を追加し, (3) 以降を繰り返す.

4.1 表現式の生成

L_{AX} を求めるために, 従来の変換法における中間表現構成法 (3.2節参照) を利用する. しかし, 本手法では各語句に対する中間表現が与えられていないという仮定をおいている. そこで, 自然語の構文情報を用いて, 引数の個数や引数に対応する自然語句を決定し, 自然語の各語句に対して, 属性 type を含まない中間表現を機械的に構成する.

例 7: 例 1で示した自然語文において, "in" を伴う "results" と "S-SYNC-MAJOR indication" に対して機械的に中間表現を構成する. "in" を伴う "results" は, 引数を 2 個とる. 第 1 引数は, 主語に対応する名詞句に対応し, 第 2 引数は, 目的語に対応する名詞句に相当する. このことから, "in" を伴う "results" に対する中間表現の根頂点に付随する C 構造は, 図 11-(1) のようになる. ここで, 関数記号は対応する自然語句から機械的に割り当てたものである. 次に, 名詞句 "S-SYNC-MAJOR indication" を考える. 一般に, 固有名詞を除く名詞は, 公理に変換されると変数に対応しており, 中間表現では, その変数が満たすべき前提条件を指定する. したがって, "S-SYNC-MAJOR indication" に対する中間表現の根頂点に付随する C 構造は, 図 11-(2) のようになる. □

次に, 従来の変換法と同様に, 入力となる自然語仕様の各文を, 自然語の構文規則を用いて構文木に変換する. この構文木にしたがって, 先に構成した語句に対する中間表現をもとに, 文に対する中間表現をボトムアップに構成する. このようにして得られた文に対する中間表現における, 属性 trans の値の集合が L_{AX} となる. ただし, 異なる文から得られた 2 つの表現式には, 同一の変数は現れないものとする.

例 8: 例 1で示した自然語文から求められる表現式の集合は, $\{\text{result_in}(\bar{x}_1, \bar{x}_2),\ \bar{x}_1,\ \bar{x}_2,\ \text{MAP}(\bar{x}_1),\ \text{valid}(\bar{x}_1),\ \text{incoming}(\bar{x}_1),\ \text{SSYNMind}(\bar{x}_2)\}$ である. □

4.2 構文規則の生成

前節で得られた L_{AX} から, $G = (N, T, P)$ を構成する. まず, N, T, P を表 4のように初期化する. 各非終端記号の直観的な意味は, 次の通りである. A_f は, 関数 f の関数値のデータタイプを表す. $A_{(f,i)}$ は, 関数 f の第 i 引数が取り得るデータタイプを表す (実引数のデータタイプと区別するため, 仮引数のデータタイプと呼ぶことがある). $A_{\bar{x}}$ は, 公理の変数 $\bar{x}$ のタイプ指定を表す.

表 4: 文法 G の初期化

$$\begin{aligned} T &= \{\text{“,”}, \text{“(”}, \text{“)”}\} \cup \{f \mid f \text{ は } L_{AX} \text{ に現れる関数記号}\} \\ N &= \{A_f \mid f \in T \text{ は } n \text{ 引数関数記号}\} \\ &\quad \cup \{A_{(f,1)}, \ldots, A_{(f,n)} \mid f \in T \text{ は } n \text{ 引数関数記号}\} \\ &\quad \cup \{A_{\bar{x}} \mid \bar{x} \text{ は } L_{AX} \text{ に現れる変数}\} \\ P &= \{A_f \to f(A_{(f,1)}, \ldots, A_{(f,n)}) \\ &\qquad \mid f \in T \text{ は } n \text{ 引数関数記号}\} \\ &\quad \cup \{A_c \to c \mid c \in T \text{ は定数}\} \end{aligned}$$

次に, G が満たすべき型包含関係として, $A_{(f,i)} \stackrel{*}{\Rightarrow} A$ (A は非終端記号) の形の条件を以下のように求める.

- L_{AX} 中に現れる変数 $\bar{x}$ と関数 f に対し, $f(\ldots, \underset{i}{\bar{x}}, \ldots)$ が L_{AX} に存在するならば, $A_{(f,i)} \stackrel{*}{\Rightarrow} A_{\bar{x}}$ は G が満たすべき型包含関係である. 直観的には, 変数 $\bar{x}$ のデータタイプは, 関数 f の i 番目の仮引数のデータタイプの部分データタイプであることを意味する.
- L_{AX} 中に現れる関数 (あるいは定数) g と関数 f に対し, $f(\ldots, \underset{i}{g(\cdots)}, \ldots)$ が L_{AX} に存在するならば, $A_{(f,i)} \stackrel{*}{\Rightarrow} A_g$ は G が満たすべき型包含関係である. 直観的には, g の関数値のデータタイプは, 関数 f の i 番目の仮引数のデータタイプの部分データタイプであることを意味する.

4.3 構文規則の簡単化

4.2節で生成された文法を G とし, G が満たすべき型包含関係の集合を R とする. ここでは, R を満たし, G と構造的に等価な最簡 (非終端記号の数および構文規則の数が最小) の文脈自由文法を構成する手法について述べる.

2 つの文脈自由文法 G_1 と G_2 が構造的に等価であるということを以下のように定義する.

定義 1: 文脈自由文法 G から生成できる構文木の集合を $TREE_G$ と書く. 任意の $tree = (V, E) \in TREE_G$ に対し, 写像 $h_V: V \to V$ と $h_E: E \to E$ を以下のように定義する.

$$h_V(v) = \begin{cases} h_V(v') & (v \text{ が唯一の子頂点 } v' \text{ をもち, かつ, } v' \text{ が内部頂点のとき}) \\ v & (\text{それ以外のとき}) \end{cases}$$

$$h_E((v_1, v_2)) = \begin{cases} (h_V(v_1), h_V(v_2)) & (h_V(v_1) \neq h_V(v_2) \text{ のとき}) \\ \text{未定義} & (h_V(v_1) = h_V(v_2) \text{ のとき}) \end{cases}$$

さらに, $h(tree) = (\{h_V(v) \mid v \in V\}, \{h_E(e) \mid e \in E\})$ とし, $h(TREE_G) = \{h(tree) \mid tree \in TREE_G\}$ と定義する.

任意の 2 つの文脈自由文法 G_1 と G_2 に対し, $h(TREE_{G_1})$, $h(TREE_{G_2})$ 中の構文木の内部頂点につけられたラベル (非終端記号) を無視したときに $h(TREE_{G_1}) = h(TREE_{G_2})$ であるならば, G_1 と G_2 は構造的に等価であるという. □

直観的には, 構文木の内部頂点のラベルを無視し, 単記号規則 ($A \to A'$) の適用を無視したとき, G_1 における構文木の全体集合が G_2 における構文木の全体集合に等しいことである.

文献 [3] では, 任意に与えられた 2 つの括弧付文脈自由文法が弱等価か否かを判定するアルゴリズムが述べられている.

このアルゴリズムを利用すると, 与えられた文脈自由文法 G に対して, 構造的に等価な非終端記号 (以下で定義) の同値類を求めることにより, G と構造的に等価な最簡の文脈自由文法を構成することができる. ここで, 2つの非終端記号が構造的に等価であるという関係を以下のように定義する.

定義 2: $TREE_G$ の元のうちで, 非終端記号 A がラベルづけされた頂点を含む構文木の集合を $TREE_G[A]$ と書く. $TREE_G[A]$ の各元から, A を根頂点とする部分木を1つ取り除いた (ただし, A をラベルとする頂点は取り除かない) 構文木の集合を $TREE_G[A \leftarrow \Omega]$ と表す. また, 写像 h を定義1と同様に定義する.

文脈自由文法 G における2つの非終端記号 A, B に対し, $h(TREE_G[A \leftarrow \Omega])$, $h(TREE_G[B \leftarrow \Omega])$ 中の構文木の内部頂点のラベルを無視したときに $h(TREE_G[A \leftarrow \Omega]) = h(TREE_G[B \leftarrow \Omega])$ であるならば, A と B は構造的に等価であるという. □

しかし, 文献 [3] で述べられているアルゴリズムの時間計算量は, 非終端記号の数 r に対して, $O(2^{2^r})$ であり, 実際に用いるアルゴリズムとしては適当ではない. 本手法で生成する文法は, 関数はプレフィックス表現 $f(t_1, \ldots, t_n)$ のみで表されているなど, 構文規則の形に特徴があるので, これを利用し, 計算時間の改善をはかる.

例えば, 生成された文法 G に対して, 以下の仮定をおく.

任意の関数記号 f に対して,

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

を満たす A は, 高々1つである.

このとき, 定義2より以下のことが示せる.

非終端記号 A, B が構造的に等価であるための必要十分条件は, A, B が任意の関数記号 f と引数位置 i に対して

$$A_{(f,i)} \overset{*}{\underset{G}{\Rightarrow}} A \text{ iff } A_{(f,i)} \overset{*}{\underset{G}{\Rightarrow}} B \tag{3}$$

を満たすことである.

本手法では, 上の仮定をおくことにより文法 G の記述長の多項式時間で最簡形を求めている. なお, 関数がプレフィックス表現された文法に限らず, 演算子とその引数の位置が一意に定まるような文法 (例えば演算子順位文法) についても, 上と同様の仮定のもとで, 非終端記号が構造的に等価であるための必要十分条件が式 (3)のようになることを示すことができる.

4.4 文法が満たすべき型包含関係の追加

一般に, 代数的仕様 $SPEC = (G, AX)$ において, G の記述の簡潔さと AX の記述の簡潔さとはトレードオフの関係にある. 例えば, MAJOR SYNC POINT SPDU を表す型 MAP_type を導入したとすると, 述語 MAP (表1参照) を導入する必要はなくなるが, MAP_type 型の表現式の構文は複雑となる. 一方, 3.2節で例示したように, セションプロトコルデータ単位を表す型 SPDU を導入して構文を簡潔に記述したとき, 述語 MAP の意味を表す公理を記述しなければならない. 4.2節で述べた方法で機械的に生成された文法 G では, 関数の各引数位置と各関数値に対してデータタイプが導入されており, 変換者にとって "G が精密化され過ぎている" 場合がある.

また, 4.3節で述べた簡単化の手法を用いても文法が十分に簡単にならない場合がある. 例えば, 1引数関数 f, g と公理の変数 $\bar{x}_1, \ldots, \bar{x}_n$ を考える. 各 i $(1 \leq i \leq n)$ について, 型包含関係 $A_{(f,1)} \overset{*}{\Rightarrow} A_{\bar{x}_i}$, $A_{(g,1)} \overset{*}{\Rightarrow} A_{\bar{x}_i}$ が機械的に生成されたとする. このとき, $A_{(f,1)}$ と $A_{(g,1)}$ が同じデータタイプを表すか, あるいは, 一方が他方の部分データタイプになっているかどうかを, 機械的に決定することはできない. なぜなら, 入力となる自然語文は, "正しい" 自然語文の全体集合の有限部分集合であり, 入力が "理想的なサンプル" となっているとは限らないからである.

以上の理由より, 変換者は, 生成された文法 G が満たすべき型包含関係の集合 R に対し, 以下のように, 型包含関係を追加する必要がある (具体例は 5節参照).

(1) 動詞に対応する関数 (中間表現の属性 trans の値) の返り値を表す非終端記号や, 名詞の前提条件を表す関数 (中間表現の属性 restriction の値) の返り値を表す非終端記号など, 本稿で採用している自然語の意味論の流儀より Bool となる非終端記号 A に対して, Bool $\overset{*}{\Rightarrow} A$ および $A \overset{*}{\Rightarrow}$ Bool を R に追加する.

(2) セションプロトコルデータ単位の種類を表す名詞など, "グループ化" できる名詞に対応する公理の変数 $\bar{x}_1, \ldots \bar{x}_n$ に対して, 新たな非終端記号 A を N に導入し, 各 i $(1 \leq i \leq n)$ について $A \overset{*}{\Rightarrow} A_{\bar{x}_i}$ を R に追加する. 直観的には, "精密化され過ぎた" データタイプ $A_{\bar{x}_1}, \ldots, A_{\bar{x}_n}$ を, A というデータタイプ名で "グループ化" することを意味する. また, この他にも, 新たに必要と思われるデータタイプ (例えば表 3で示したデータタイプ) を表す非終端記号を N に導入し, これらの間の階層関係を表す型包含関係を R に追加する.

(3) 関数記号 f, 引数位置 i と変換者が導入した非終端記号 A に対し,「データタイプ A の表現式はすべて f の第 i 引数として現れうる」とき, $A_{(f,i)} \overset{*}{\Rightarrow} A$ を R に追加する. 逆に,「f の第 i 引数として現れうるのはデータタイプ A の表現式のみである」とき, $A \overset{*}{\Rightarrow} A_{(f,i)}$ を R に追加する. また, 関数記号 f, g と引数位置 i, j に対し, 入力の自然語仕様において「f に対応する語句の第 i 引数の位置に現れうる語句は, g に対応する語句の第 j 引数の位置に現れてもよい」という関係が成り立っているとき, $A_{(g,j)} \overset{*}{\Rightarrow} A_{(f,i)}$ を R に追加する.

型包含関係を追加した後に簡単化を行なう操作を繰り返す. 変換者が適切であると判断した時点で, 文法 G およびプリミティブなデータタイプに関する文法 G_0 を出力し, 辞書項目として辞書に登録する. 文法 G を出力する際には, G の満たす型包含関係 $A \overset{*}{\Rightarrow} A'$ を, $A \rightarrow A'$ の形の構文規則に変換して出力する.

5 構文規則生成支援システム

現在, 本手法に基づく構文規則生成支援システムを試作している. システムの概要を図 12に, その規模を表 5に示す.

また, このシステムに対し, 文献 [2] の内, プロトコル機械 SPM の動作を規定した部分 (29段落 98文) から, 語 "result",

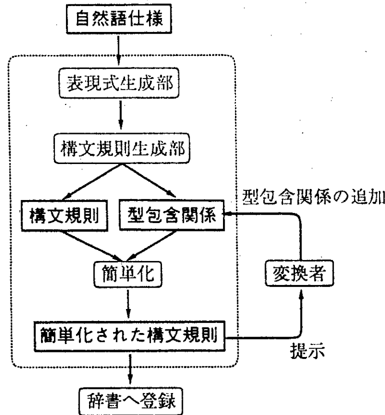

図 12: 構文規則生成支援システムの構成

表 5: 構文規則生成支援システムの規模

表現式生成部	構文木生成	構文規則を DCG[4] で記述. 規則数は約 200.
	表現式生成	prolog で記述. 節数は約 500.
構文規則生成部		prolog で記述. 節数は約 160.
簡単化		C 言語で記述. C のソースは約 230 行.

表 6: 実行結果

	result を含む文	wait を含む文	レジスタ操作を規定した文
文数	27	10	11
機械的に生成された			
構文規則の数	68	39	34
型包含関係の数	204	165	122
Nの数	171	143	106
変換者が追加した			
型包含関係の数	81	49	46
Nの数	10	5	4
簡単化後の			
型包含関係の数	27	29	20
Nの数	25	29	24
実行時間	25 sec	15 sec	12 sec

"wait" を含む文と, SPM のレジスタの操作を規定している文を入力として与えた. このときの実行結果を表 6に示す. ただし, $A \overset{*}{\Rightarrow} A'$ および $A' \overset{*}{\Rightarrow} A$ の形の 2 つの型包含関係を同時に追加するものについては, 表 6では 1 つの型包含関係として数えている. また, 本システムでは, 名詞の前提条件を表す関数 (属性 restriction の値) の返り値を表す非終端記号はすべて自動的に`Bool` に置き換えるようになっており, これらの置き換えに相当する型包含関係の数は表 6には含まれていない. 実行時間の計測は, SPARCstation IPX で行なった.

例 9: 語 "result" を含む 27 文の中の 5 文を表 7に示す. この 5 文を例として, 語 "result" に対する構文宣言を生成する場合を考える. 表 7の 5 文をシステムへ入力すると, 出力として構文規則数 15, 型包含関係数 29, 非終端記号数 31 が得られる. その一部を表 8に示す. システムからの出力では, 4.2 節での表記に従い, 機械的に生成された非終端記号を`A[関数記号]` や`A[関数記号, 引数位置]` のように表す.

表 7: システムへの入力例

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

表 7の文からの出力に対して, 4.4節で述べた手順にしたがって, 以下のように型包含関係を追加する.

(1) 名詞の前提条件を表す関数の返り値を表す非終端記号以外で`Bool` と判断できる`A[result_in]` に対して, 型包含関係 "`A[result_in] <==> Bool`" を追加する. ここで, "`A <==> B`" は, 「$A \overset{*}{\Rightarrow} B$ かつ $B \overset{*}{\Rightarrow} A$」を表す.

(2) 文献 [2] では, セションプロトコルデータ単位やセションサービスプリミティブとしてどのようなものがあるかが記述されている. その記述に従い, 表 9に示した型包含関係を追加する. 表における "`A ===> B1|...|Bn`" は, 「$A \overset{*}{\Rightarrow} B1, \ldots, A \overset{*}{\Rightarrow} Bn$」を表す. ここでは, "ABORT ACCEPT SPDU" や "ABORT SPDU" など, セションプロトコルデータ単位を表す名詞に対応する公理の変数を, 新たなデータタイプ`SPDU` でグループ化している. 同様に, セションサービスプリミティブを表す名詞についても, 新たなデータタイプ`SSprm` を導入してグループ化している. また, 新たに導入したデータタイプに対して, 表 10に示した型包含関係を追加する. ここでは, 表 3で示したデータタイプを導入し, それらの間の階層関係を図 2にしたがって記述している.

(3) 表 11に示した型包含関係を追加する. 1 番目の型包含関係では, 「`abort_accept_spdu` の第 1 引数として現れうるのは, ちょうどデータタイプ`SPDU` の表現式である」ことを指定している. また, 9 番目の型包含関係では, 「入力の自然語仕様において, "assignment" の第 1 引数として現れうる語句はちょうど "transport connection" だけである」ことを指定している.

このように型包含関係を加え, 簡単化を行なった結果を表 12に示す. 表 12における`nt*`は, 構造的等価性に基づく非終端記号の同値類に, 機械的に割り当てられた名前である. この同値類を表 13に示す. このようにして得られた構文規則に対し, 同値類の中から代表元を選ぶことにより, 非終端記号の名前を付け換えたものを, 表 14に示す. □

6 あとがき

本稿では, 入力となる自然語仕様を, 自然語の構文情報をもとに解析することにより, 公理に現れる表現式の構文規則を, 機械的に生成する手法について述べた. これにより, 自然語仕様から代数的仕様への変換で用いられる辞書を作成するための労力を軽減することができる.

表 8: (**S5**) に対するシステムの出力例

- 構文規則

```
A[result_in] ---> result_in(A[result_in,1],A[result_in,2])
Bool ---> incoming(A[incoming,1])
Bool ---> valid(A[valid,1])
Bool ---> major_sync_point_spdu(A[major_sync_point_spdu,1])
Bool ---> s_sync_major_indication(A[s_sync_major_indication,1])
```

- 型包含関係

```
A[s_sync_major_indication,1] ===> A[_x9]
A[result_in,2] ===> A[_x9]
A[major_sync_point_spdu,1] ===> A[_x10]
A[result_in,1] ===> A[_x10]
A[valid,1] ===> A[_x10]
A[incoming,1] ===> A[_x10]
```

表 9: "グループ化" を表す型包含関係

```
SPDU  ===> A[_x3] | A[_x6] | A[_x7] | A[_x8] | A[_x10]
SSprm ===> A[_x1] | A[_x4] | A[_x5] | A[_x9]
```

表 10: データタイプの階層を表す型包含関係

```
In_data ===> In_SPDU | In_SSprm
In_event ===> In_data
Out_data ===> Out_SPDU | Out_SSprm
Out_event ===> Out_data
SPDU ===> In_SPDU | Out_SPDU
SSprm ===> In_SSprm | Out_SSprm
```

表 11: 関数の引数に関する型包含関係

```
A[abort_accept_spdu,1] <==> SPDU
A[abort_spdu,1] <==> SPDU
A[accept_spdu,1] <==> SPDU
A[major_sync_point_spdu,1] <==> SPDU
A[s_connect_accept_confirm,1] <==> SSprm
A[s_connect_accept_response,1] <==> SSprm
A[s_connect_request,1] <==> SSprm
A[s_sync_major_indication,1] <==> SSprm
A[assignment,1] <==> A[transport_connection,1]
A[incoming,1] <==> In_SPDU
A[result_in,1] <==> In_event
A[result_in,2] <==> Out_event
A[sending,1] <==> SPM
A[sending,2] <==> SPDU
A[valid,1] <==> SPDU
```

表 12: 簡単化された構文規則

```
nt1 ---> abort_accept_spdu(nt5)
nt1 ---> abort_spdu(nt5)
nt1 ---> accept_spdu(nt5)
nt1 ---> incoming(nt7)
nt1 ---> major_sync_point_spdu(nt5)
nt1 ---> result_in(nt8,nt9)
nt1 ---> s_connect_accept_confirm(nt10)
nt1 ---> s_connect_accept_response(nt10)
nt1 ---> s_connect_request(nt10)
nt1 ---> s_sync_major_indication(nt10)
nt1 ---> transport_connection(nt6)
nt1 ---> valid(nt5)
nt5 ---> nt3 | nt7
nt8 ---> nt2 | nt7
nt9 ---> assignment(nt6)
nt9 ---> sending(nt11,nt5)
nt9 ---> nt3 | nt4
nt10 ---> nt2 | nt4
```

表 13: 非終端記号の同値類

```
nt1  = {A[result_in],Bool}
nt2  = {A[_x2],A[_x4],In_SSprm}
nt3  = {A[_x3],Out_SPDU}
nt4  = {A[_x5],A[_x9],Out_SSprm}
nt5  = {A[_x7],A[abort_accept_spdu,1],A[abort_spdu,1],
        A[accept_spdu,1],A[major_sync_point_spdu,1],A[sending,2],
        A[valid,1],SPDU}
nt6  = {A[_x1],A[assignment,1],A[transport_connection,1]}
nt7  = {A[_x6],A[_x8],A[_x10],A[incoming,1],In_SPDU}
nt8  = {A[result_in,1],In_data,In_event}
nt9  = {A[assignment],A[result_in,2],A[sending],Out_data,
        Out_event}
nt10 = {A[s_connect_accept_confirm,1],
        A[s_connect_accept_response,1],A[s_connect_request,1],
        A[s_sync_major_indication,1],SSprm}
nt11 = {A[sending,1],SPM}
```

表 14: 名前を付け換えた構文規則

```
Bool ---> abort_accept_spdu(SPDU)
Bool ---> abort_spdu(SPDU)
Bool ---> accept_spdu(SPDU)
Bool ---> incoming(In_SPDU)
Bool ---> major_sync_point_spdu(SPDU)
Bool ---> result_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 ---> transport_connection(Tcon)
Bool ---> valid(SPDU)
Out_event ---> assignment(Tcon)
Out_event ---> sending(SPM,SPDU)
Out_event ---> Out_SPDU | Out_SSprm
In_event ---> In_SPDU | In_SSprm
SPDU ---> In_SPDU | Out_SPDU
SSprm ---> In_SSprm | Out_SSprm
```

今後は, 本稿で示した生成支援システムにおいて, 仕様記述者が制約条件を追加する労力を軽減するために, 機械的に生成可能なより強い制約条件を考察する予定である.

参考文献

[1] 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., **E76-D**, 12 (to appear).

[2] ISO: "Basic connection oriented session protocol specification", ISO 8327.

[3] McNaughton, R.: "Parenthesis Grammar", Journal of ACM, **14**, 3, pp.490–500, (1967–07).

[4] Pereira, F. C. N. and Warren, D. H. D., "Definite Clause Grammars for Language Analysis", Artificial Intelligence, **13**, pp. 231–278, (1980).

[5] 大崎敦司, 石原靖哲, 関浩之, 嵩忠雄: "自然語仕様から代数的仕様への変換における辞書項目生成の支援", 情報処理学会第46回全国大会, 4A-1 (1993–03).

[6] 嵩忠雄, 谷口健一, 杉山裕二, 関浩之: "代数的言語 ASL/* — 意味定義を中心に —", 信学論 (D), **J69-D**, 7, pp. 1066–1074 (1986–07).

[7] 関浩之, 嵩忠雄, 並河英二, 松村享: "自然語仕様から代数的仕様への変換法について", 信学論 (D-I), **J74-D-I**, 4, pp. 283–295 (1991–04).

[8] 東野輝夫, 関浩之, 谷口健一: " 代数的仕様から関数型プログラムの導出とその実行", 情報処理, **29**, 8, pp. 881–896 (1988–08).