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

4A−1

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

1 はじめに

筆者らは, 自然語仕様から代数的仕様への変換法に関する研究を行っており, 変換システムも試作している [1, 3]. 代数的仕様 t とは, 文脈自由文法 G と公理の集合 AX との2字組 $t=(G,AX)$ である. G によってデータタイプの宣言や関数の構文宣言 (関数名および引数と関数値のデータタイプの宣言) を行う. G で導入された関数の意味は, 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.

自然語仕様における変換の最小単位を, 語句と呼ぶ. 本変換法では, 自然語の各段落を多ソートの一階述語論理式の形の公理に変換する. 変換された代数的仕様におけるデータタイプの宣言や関数の構文宣言は, 自然語仕様に現れる語句に対する辞書項目から構成される.

試作システムにおける辞書項目は, 自然語仕様を理解した人間によって作成されている. しかし, データタイプを適切に導入・階層化するのは労力を必要とする作業である. 本稿では, 入力となる自然語仕様を解析することによって各語句に対応する関数の構文宣言の記述を支援する手法について, 文献 [2] を例として考察する.

2 構文の精密化によるあいまいさの除去

公理に現れる式のデータタイプや構文の宣言部は, 前節で述べたように不可欠なものであると同時に, 適切なデータタイプを導入することによって, 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 1: 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.

さらに, "ABORT SPDU" と "ABORT ACCEPT SPDU" に対応する式のデータタイプがSPDU型であることがわかっているとする. しかし, これだけでは, "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, プロトコル機械に与えられる事象を表す型 (*in_event*型) とプロトコル機械が起こす事象を表す型 (*out_event*型) を導入し, *in_event*型の式の集合と*out_event*型の式の集合は互いに素であるとする. さらに, 受信されるデータを表すデータタイプ (*in_data*型) と送信されるデータを表すデータタイプ (*out_data*型) を, それぞれ, *in_event*型, *out_event*型の部分データタイプとして定義する. 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, 目的語に対応する式のデータタイプは, それぞれ, *in_event*型, *out_event*型である.
- "be sent on" の主語に対応する式のデータタイプは*out_data*型である.

これにより, "this SPDU" に対応する式のデータタイプは*out_data*型となり, "results in" の主語である "ABORT SPDU" を照応し得ないと決定できる. □

表 1: 中間表現の属性

属性	属性値
trans	自然語文または句の意味を表す論理式
type	属性 trans で指定された論理式の代数的仕様におけるデータタイプ
args	属性 trans で指定される論理式が関数記号 (または述語記号) のとき, その引数に関する情報
subcat	その中間表現が対応づけられた自然語文または句を SC とすると, SC の部分句に対応づけられる中間表現に関する情報
restriction	属性 trans で指定される論理式が変数の時, その変数の満たすべき前提条件

3 分析法

まず, 自然語仕様を形式的に分析するための概念として, 中間表現を導入する. 中間表現とは, <属性, 値>の2字組の集合であり, ここでは表1に示す属性を用いる.

入力仕様中に現れる語句 word に対する中間表現を次のように機械的に生成する. word に対応する論理式が関数 $word(X_1, X_2, \ldots, X_n)$ であるとき, 第 1, 2,..., n 引数のデータタイプをそれぞれ, word(*,...,), word(,*,...,),..., word(,...,*) とする. さらに,

- word が動詞または名詞化した動詞の場合: 関数 *word* のデータタイプを word_type とする. 例えば, "result in" に対して生成される中間表現は次のようになる.

$$\left[\begin{array}{ll} \text{trans} & result\_in(X,Y) \\ \text{type} & \text{result\_in\_type} \\ \text{subcat} & \left[\left[\begin{array}{ll} \text{trans} & X \\ \text{type} & \text{result\_in}(*,) \end{array}\right], \left[\begin{array}{ll} \text{trans} & Y \\ \text{type} & result\_in(,*) \end{array}\right]\right] \end{array}\right]$$

Translation from Natural Language Specifications into Algebraic Specifications — Support to Generation of Lexical Items —
Atsushi Ohsaki†, Yasunori Ishihara†, Hiroyuki Seki†, and Tadao Kasami‡†
† Dept. of Information & Computer Sciences, Osaka University
‡ Advanced Institute of Science and Technology, Nara

表 2: 文献 [2] における文の例

(S1) A valid incomming ACCEPT SPDU results in an S-CONNECT (accept) confirm.

(S2) An S-CONNECT request results in the assignment of a transport connection.

(S3) An S-CONNECT (accept) response results in an ACCEPT SPDU.

(S4) An ACCEPT SPDU is sent on the transport normal flow.

(S5) A valid incomming ABORT SPDU results in sending an ABORT ACCEPT SPDU.

表 3: 表2の文より求まるデータタイプ

(S1) T_1 = valid(*) ∧ incomming(*) ∧ AC(*) ∧ result_in(*,)
T_2 = SCONcnf+(*) ∧ result_in(,*)

(S2) T_3 = SCONreq(*) ∧ result_in(*,)
T_4 = assignment_type ∧ result_in(,*)

(S3) T_5 = SCONrsp+(*) ∧ result_in(*,)
T_6 = AC(*) ∧ result_in(,*)

(S4) T_7 = AC(*) ∧ be_sent_on(*,)
T_8 = trans_norm_flow(*) ∧ be_sent_on(,*)

(S5) T_9 = valid(*) ∧ incomming(*) ∧ AB(*) ∧ result_in(*,)
T_{10} = sending_type ∧ result_in(,*)
T_{11} = AA(*) ∧ sending(*)

- word がそれ以外の語句の場合: 関数 *word*のデータタイプを bool とする. 例えば, "S-CONNECT (accept) confirm" (SCONcnf+と略記する) に対して生成される中間表現は次のようになる.

$$\left[\begin{array}{ll} \text{trans} & X \\ \text{type} & \text{SCONcnf+(*)} \\ \text{restriction} & \left[\left[\begin{array}{ll} \text{trans} & \textit{SCONcnf+}(X) \\ \text{type} & \text{bool} \end{array}\right]\right] \end{array}\right]$$

このようにして, 入力仕様中に現れるすべての語句に対する中間表現を生成する. そして, 文に対する中間表現をボトムアップに構成する. 例えば, "result in an SCONcnf+" に対する中間表現は, "result in" に対する中間表現における属性 subcat の末尾の中間表現, および "SCNcnf+" の中間表現を単一化 (二つの中間表現の情報を合わせ持つような中間表現を作る操作) した結果を, subcat から args に移動したものであると定義され, この場合は以下のようになる.

$$\left[\begin{array}{ll} \text{trans} & \textit{result\_in}(X,Y) \\ \text{type} & \text{result\_in\_type} \\ \text{args} & \left[\left[\begin{array}{ll} \text{trans} & Y \\ \text{type} & \text{SCONcnf+(*)} \wedge \text{result\_in(,*)} \\ \text{restriction} & \left[\left[\begin{array}{ll} \text{trans} & \textit{SCONcnf+}(Y) \\ \text{type} & \text{bool} \end{array}\right]\right] \end{array}\right]\right] \\ \text{subcat} & \left[\left[\begin{array}{ll} \text{trans} & X \\ \text{type} & \text{result\_in(*,)} \end{array}\right]\right] \end{array}\right]$$

ここで, $A \wedge B$ は, A と B 両方の部分データタイプの中で最も一般的なデータタイプを表す. 表 2の各文に対する中間表現における属性 args の中の属性 type の値を求めると, 表 3のようになる.

このようにして得られたデータタイプの間の包含関係を, 次のような有向グラフで表現する.

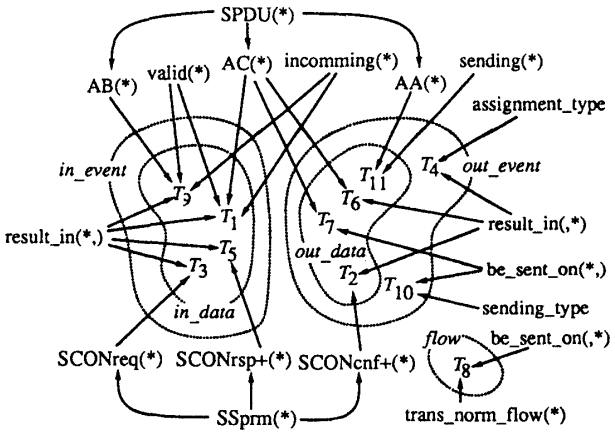

図 1: データタイプ間の包含関係を表す有向グラフ

result_in_type → result_in(*in_event*, *out_event*)
be_sent_on_type → be_sent_on(*out_data*, *flow*)
in_event → *in_data*
out_event → *out_data*

図 2: 構文宣言の例

- 頂点の集合はデータタイプの集合と一対一対応している (以下, 頂点とデータタイプを同一視する).
- データタイプ t' が t の部分データタイプであるときかつそのときのみ, t から t' への道が存在する.

表3に示したデータタイプに対してこの有向グラフを構成したものが, 図1である. 頂点 SPDU(*) および SSprm(*) は, 入力仕様中の表で宣言されている語句である, SPDU, SS-primitive に対応する頂点である.

このグラフをもとに, 仕様記述者は, 代数的仕様で用いるデータタイプを選択し, 各関数の構文宣言を記述して辞書に格納する. このとき, 必要ならば, 頂点の部分集合に新たなデータタイプを割り当てる. 図 1では, *in_data*型, *out_data* 型, *in_event*型, *out_event*型および *flow* 型を導入している. 図 1をもとに記述した構文宣言の例を図 2に示す.

参考文献

[1] Ishihara, Y., Seki, H. and Kasami, T., "A Translation Method from Natural Language Specifications into Formal Specifications Using Contextual Dependencies", Proc. of IEEE International Symposium on Requirements Engineering '93, pp. 232–239 (1993–01).

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

[3] 関, 嵩, 並河, 松村: "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).